Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo Fin de Grado

ESTUDIO DE ALGORITMOS DE COMPUTACIÓN BASADA EN GRAFOS APLICADO AL ANÁLISIS DE REDES

Autor: Pinar Sanz Sacristán
Tutor: David Camacho Fernández

Enero 2018

ESTUDIO DE ALGORITMOS DE COMPUTACIÓN BASADA EN GRAFOS APLICADO AL ANÁLISIS DE REDES

Autor: Pinar Sanz Sacristán
Tutor: David Camacho Fernández

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero 2018

Resumen

Resumen

Hoy en día, todo el mundo usa las redes sociales para compartir todo tipo de información e interactuar con otras personas. Los datos de las redes sociales pueden modelarse fácilmente mediante grafos. Esta representación permite que se pueda extraer información útil de la red mediante la aplicación de algoritmos. En particular, con el objetivo de identificar grupos de individuos con gustos, intereses o ideas similares, existen los algoritmos de detección de comunidades.

En este trabajo se realiza un estudio de una serie de algoritmos populares de detección de comunidades del estado del arte. La red social de la cual se ha obtenido el conjuntos de datos, sobre el cual se van a probar los algoritmos, es Twitter. El volumen de gente que usa esta red social y la facilidad que tiene de publicar y reenviar mensajes (a través de *retweets*) hacen que la información se propague con rapidez, llegando a muchos usuarios. Por esta razón es utilizada, en particular, por radicales del ISIS con el objetivo de reclutar gente. Este tipo de perfiles son los que componen el conjunto de datos que se va a utilizar para la ejecución de los algoritmos de detección de comunidades.

Para el estudio de la red se han obtenido las redes ego que la componen. Una red ego está formada por un nodo central y todos sus vecinos y los enlaces que conectan a estos. Este tipo de redes puede resultar interesante con el conjunto de datos utilizado ya que es común que exista un individio que trate de adoctrinar a otros, por lo que sería el representado como *ego*. Por otro lado, se ha planteado el uso de modelos de datos alternativos al popular *following-follower*. Estos modelos se basan en tener en cuenta las menciones a usuario, respuestas a *tweets* entre usuarios o *retweets*.

Los resultados que se han obtenido revelan que los modelos alternativos obtienen buenos resultados, especialmente con algunos algoritmos. Las comunidades que se han obtenido tienen una conductancia y separabilidad bajas y una cohesividad alta, que son las características que indican que una comunidad es fácilmente identificable. Además, han obtenido mejores resultados que el modelo corriente *following-follower*.

Palabras Clave

Algoritmos de detección de comunidades, Redes ego, Redes, Grafos, Twitter

Abstract

Nowadays, everybody uses Social Networks to share all kind of information and interact with other people. Social networks data can be easily modeled by graphs. This representation allows the extraction of useful information from the network through the application of algorithms. In particular, with the aim of identifying groups of individuals with similar tastes, interests or ideas, community detection algorithms exist.

In this work a series of popular State-of-the-Art algorithms in Community Finding are studied. The social network from which the dataset has been obtained, on which the algorithms will be tested, is Twitter. The volume of people who use this social network and the ease of publicating and re-sending messages (through retweets) make the information spread quickly, reaching many users. For this reason it is used, in particular, by ISIS radicals with the aim of recruiting people. These types of profiles are those that compose the data set that will be used for the execution of the community detection algorithms.

For the study of the network, the ego networks that compose it have been obtained. An ego network is formed by a central node and all its neighbors and the links that connect them. This type of networks may be interesting with the dataset used since it is common for there to be an individual who tries to indoctrinate others, so he would be represented as the ego. On the other hand, the use of data models alternative to the typical following-follower has been proposed. These models are based on taking into account user mentions, responses to tweets between users or retweets.

The obtained results reveal that the alternative models obtain good results, especially with some algorithms. The obtained communities have low conductance and separability and high cohesiveness, which are the characteristics that indicate that a community is easily identifiable. In addition, they have obtained better results than the current model following-follower.

Key words

Community Detection Algorithms, Ego Networks, Networks, Graphs, Twitter

Agradecimientos

Este trabajo supone el final de una importante etapa de mi vida y tengo que agradecer a todos aquellos que me han ayudado a llegar hasta aquí.

En primer lugar a David, que no solo me ha dirigido este trabajo, siempre me ha ayudado a encontrarme cada vez que me he perdido y me introdujo en el grupo AIDA. Este grupo también merece mención, ya que me han ayudado siempre que se lo he pedido, especialmente Gema, Antonio, y Víctor, que será de las pocas personas que se lean mi trabajo. Y gracias a Irene y Javi por hacer del despacho un lugar mejor durante todo este tiempo.

A mi familia, que siempre me han apoyado, hasta el punto de intentarse leer mi trabajo a pesar de no saber lo que es un grafo (creo que ya sí). Y destaco a mi madre, que siempre ha sido mi mejor amiga y consejera.

Y a Mateo, que me ha acompañado durante todo este camino y siempre ha sido mi apoyo. Sin él no sé si habría llegado hasta el final.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

Actualmente vivimos en un mundo conectado, en el cual se han creado diferentes redes sociales, SNs (*Social Networks*) que unen a millones de personas de todas partes del mundo. En ellas la gente interactúa, comparte fotos, vídeos, *posts*... Con ello, surge el interés en estudiar cómo se relacionan las personas, cómo interactúan entre sí. De este interés, surge el análisis de redes sociales, SNA (Social Network Analysis), que se encarga de estudiar la estructura social que se forma mediante el uso de redes y teoría de grafos.

Las redes pueden estudiarse a través de las características o propiedades de sus elementos, o mediante la estructura que se forma debido a las interacciones de los elementos. Este trabajo va a centrarse en el análisis de la estructura de la red mediante el problema de detección de comunidades, CDP (*Community Detection Problem*).

Todo esto no solo se puede aplicar para el análisis de redes sociales, si no que también se aplica en otros dominios como la sociología, el *marketing*, la biología o la neurociencia. Por ejemplo, en [7] lo utilizan para analizar los votos del Festival de Eurovisión y, en [8], hacen un análisis de las redes de coautores en artículos de investigación.

En este capítulo se describe cuál es la motivación del proyecto. Después se muestran los objetivos y el plan de trabajo a llevar a cabo. Finalmente se presentará la estructura del resto del documento.

1.1. Motivación del proyecto

Los algoritmos enfocados a la detección de comunidades se encargan de dividir un grafo que representa la red en *clusters* o comunidades. En este trabajo se comparará el resultado de una serie de algoritmos clásicos enfocados a la detección de comunidades mediante varias métricas.

Debido a la gran cantidad de información que puede obtenerse en las redes sociales, es común aplicar enfoques basados en el paradigma del *Big Data* [9]. Otras medidas que se han tomado para evitar analizar toda la red a la vez se basan en reducir la cantidad de datos a estudiar. Una de estas medidas, que será la que se utilizará para analizar los datos en este trabajo, es dividir la red en las redes ego que la forman. Esta aproximación se usa frecuentemente para este tipo de problemas [10, 11, 12].

Una red ego es un subgrafo de una red social compuesto por un nodo central, el denominado *ego*, los usuarios conectados a él (*alters*) y las relaciones entre los *alters*.

El dominio en el que se van a probar los algoritmos está compuesto por perfiles de radicalizados relacionados con el ISIS (*Islamic State in Iraq and Syria*). Este conjunto de datos resulta interesante ya que es un tema que ha ganado relevancia en los últimos años, debido a los atentados terroristas ocurridos recientemente. Este grupo utiliza las redes sociales no solo para difundir sus ideas, si no también para reclutar gente que se una a ellos. El conjunto de datos que se va a utilizar incluye perfiles publicados por Anonymous y Kaggle y no está contrastado.

La red social de la que se han extraído los perfiles es Twitter, una de las redes sociales más populares y utilizadas a diario. Es un servicio de *microblogging* fundada en 2006, en la cual los usuarios pueden escribir mensajes, llamados *tweets*, de 280 caracteres. Por defecto, las cuentas son públicas por lo que cualquiera puede leer los *tweets* que se publican e interactuar con ellos, lo que hace que sea más fácil difundir ideas y generar más contenido.

1.2. Objetivos y plan de trabajo

Con este trabajo se pretende estudiar el resultado de diferentes algoritmos basados en grafos aplicados a un dominio de datos concreto, y comparar la viabilidad de un nuevo modelo de representación. Normalmente se ha utilizado un modelo *following/follower*, y en este trabajo se pretende ver la efectividad de aplicar un modelo en el que se incluyan menciones, respuestas y *retweets*. Esto quiere decir que dos usuarios estarán conectados, es decir, existirá un enlace entre ellos, si han interactuado de alguna manera: uno a mencionado, respondido o *retweeteado* al otro.

El proceso seguido es el siguiente:

1. **Creación de la base de datos**: representación de los datos mediante Neo4j [13], una base de datos orientada a grafos.
2. **Preprocesado**: obtención de los subgrafos a estudiar (redes ego) de la base de datos y almacenamiento de los mismos en formato *graphml*.
3. **Ejecución**: mediante la librería Circulo [14] se realizarán los siguientes pasos:
    1) Ejecución de algoritmos: los algoritmos del estado del arte se ejecutan contra los datos que se han preprocesado en el paso anterior.
    2) Ejecución de métricas: el resultado de la ejecución de los algoritmos es evaluada contra una variedad de métricas.
4. **Estudio de resultados**: para ello se van a tener en cuenta tres métricas: conductancia, cohesividad y separabilidad. Esta información se obtiene de la salida de Circulo mediante un *script*. De esta manera es más fácil estudiar los resultados obtenidos.
5. **Representación gráfica**: por último se representarán algunos ejemplos de redes divididas en comunidades según distintos algoritmos.

1.3. Estructura del documento

La estructura del documento es la siguiente:

- **Capítulo 1: Introducción.** Muestra una breve introducción sobre el tema que se va a tratar en este trabajo, la motivación y los objetivos que tiene y el proceso seguido para su desarrollo.
- **Capítulo 2: Estado del arte.** Comienza definiendo algunos conceptos básicos de teoría de grafos, necesarios para la posterior explicación de los algoritmos que se van a analizar. Por último se explican algunos términos de Twitter y la API de Twitter, mediante la cual se han obtenido los datos que se usarán en el capítulo de experimentación.
- **Capítulo 3: Modelos de representación de las redes sociales.** Muestra el modelo estándar que suele utilizarse para el análisis de redes sociales y se explica el modelo alternativo. Los grafos resultantes de aplicar este modelo son los que se van a analizar.
- **Capítulo 4: Arquitectura.** En este capítulo se explican las herramientas que se han utilizado, las modificación que ha habido que realizar en ellos y el sistema que se ha desarrollado para el estudio que se lleva a cabo en este trabajo.
- **Capítulo 5: Experimentos realizados y resultados.** Primero se describen con más detalle los conjuntos de datos que se van a utilizar y, después, se exponen los resultados obtenidos junto con las conclusiones que se pueden extraer. Finalmente se muestra gráficamente algunos ejemplos de redes divididas en comunidades según distintos algoritmos.
- **Capítulo 6: Conclusiones y trabajo futuro.** Finalmente, se comentan los resultados que se pueden obtener tras la realización de este proyecto y las posibles ampliaciones que se podrían hacer en el futuro como continuación de este trabajo.

2

Estado del arte

Este capítulo comienza con algunas definiciones básicas relacionadas con la teoría de grafos que se utilizarán en el resto del documento. Después se explican los algoritmos de detección de comunidades que serán objeto de estudio. Finalmente, se definen términos propios de Twitter, además de comentar el funcionamiento de su API.

2.1. Conceptos básicos de teoría de grafos

El modelo de grafos es el más adecuado para la representación de datos relacionados con redes sociales, ya que se pueden representar fácilmente los usuarios como nodos y sus relaciones como enlaces. Puesto que es el utilizado en este trabajo, a continuación se van a definir algunos conceptos básicos.

Definición 2.1.1 *Un **grafo** $G = (V, E)$ es un conjunto de vértices o nodos $V = \{v_1, ..., v_n\}$ y de enlaces E, un conjunto de pares de nodos, donde cada enlace se denota por e_{ij}, si existe una relación entre los nodos v_i y v_j. En la figura 2.1 pueden verse dos representaciones de grafos.*

Figura 2.1: Ejemplos de grafos: (a) Grafo no dirigido. (b) Grafo dirigido

Definición 2.1.2 *Un grafo es **dirigido** cuando los enlaces tienen sentido (representado por una flecha). En estos grafos un nodo v_i puede estar conectado con v_j pero no viceversa. Un grafo es **no dirigido** si sus enlaces no tienen sentido, es decir, cuando existe un enlace e_{ij} entre dos nodos quiere decir que v_i y v_j están conectados en ambos sentidos. En la figura 2.1 se puede ver cómo se representa cada uno.*

Si se quiere tener en cuenta el coste de cada enlace es necesario definir un nuevo tipo de grafo:

Definición 2.1.3 *Un grafo G es **ponderado** si existe una función $w : E \to \mathbb{R}$ que asocia cada enlace con un número real.*

Definición 2.1.4 *Dos nodos v_i y v_j son **vecinos** en un grafo $G = (V, E)$ si existen los enlaces e_{ij} y e_{ji} tales que $e_{ij}, e_{ji} \in E$. En la figura 2.1 (a) puede verse que los nodos 1 y 2 son vecinos, en cambio, 1 y 3 o 1 y 5 no lo son.*

Definición 2.1.5 *El **grado** de un nodo es el número de enlaces que inciden en el nodo, es decir, que terminan en él.*

A la hora de respresentar un grafo, la manera más común de hacerlo es a través de su matriz de adyacencia, que se define como:

Definición 2.1.6 *La **matriz de adyacencia** de un grafo G=(E,V), A_G, es una matriz cuadrada $n \times n$ (n es el número de nodos de G), donde cada coeficiente a_{ij} vale 1 si existe un enlace $e_{ij} \in E$ y 0 en otro caso.*

Dentro de los grafos pueden diferenciarse estructuras que pueden ser útiles para la detección de comunidades. Algunos ejemplos de estas son los k-cliques y las componentes conexas que se definen a continuación.

Definición 2.1.7 *Un **k-clique** es un subconjunto de k nodos, todos ellos conectados dos a dos por un enlace.*

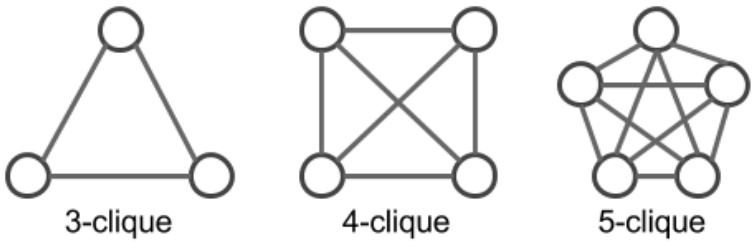

Figura 2.2: Ejemplos de 3-clique, 4-clique y 5-clique

Definición 2.1.8 *Una **componente conexa** de un grafo es un subgrafo en el cual cada par de vértices está conectado por un camino. Un grafo formado por una sola componente conexa se dice conexo.*

Definición 2.1.9 *Un **paseo** entre dos vértices v_r, v_s es una secuencia de vértices y enlaces. Puede ocurrir que $v_r = v_s$.*

Definición 2.1.10 *Un **camino** entre dos vértices v_r, v_s es una secuencia de vértices y enlaces no repetidos que conectan una serie de vértices $v_1, ..., v_k, ...v_n$ en la cual $v_1 = v_r$ y $v_n = v_s$.*

Definición 2.1.11 *Si G es conexo, el camino de menor longitud entre un vértice u a otro vértice v en G se denomina **camino más corto** (en inglés shortest path, SP) [15].*

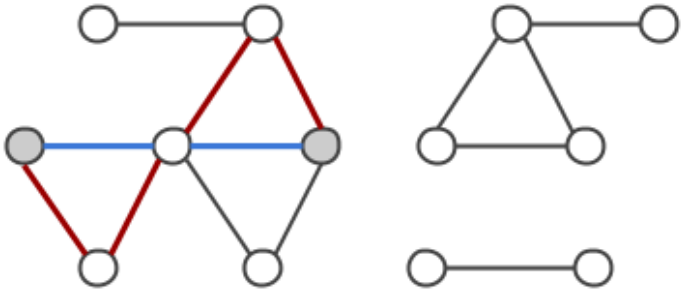

Figura 2.3: Ejemplo de grafo con tres componentes conexas. En la componente conexa de la izquierda aparecen coloreados dos posibles caminos entre los dos nodos sombreados. El camino azul es el más corto (SP) entre los dos nodos

En la figura 2.3 aparece un grafo formado por 3 componentes conexas. En la componente conexa de la izquierda aparecen coloreados dos posibles caminos entre los dos nodos sombreados, de los cuales el azul representa el camino más corto, SP, entre los nodos sombreados.

Otro término que aparecerá en la descripción de los algoritmos es centroide, pero para definirlo es necesario conocer antes el concepto excentricidad:

Definición 2.1.12 *La **excentricidad** de un vértice v es la distancia de v al vértice más lejano de v, siguiendo caminos de longitud mínima, en el grafo [15].*

Definición 2.1.13 *El **centroide** de un grafo es el conjunto de vértices con excentricidad mínima [15], es decir, el conjunto de vértices u tales que su distancia $d(u, v)$ a el resto de vértices v es minimal.*

El estudio de grafos de grandes dimensiones se puede hacer a través de subgrafos. Una criterio con el que extraer los subgrafos es la obtención de las redes ego:

Definición 2.1.14 *Una **red ego** (ego network) es un subgrafo formado por un nodo, el ego, y todos sus vecinos, denominados alters y los enlaces entre estos.*

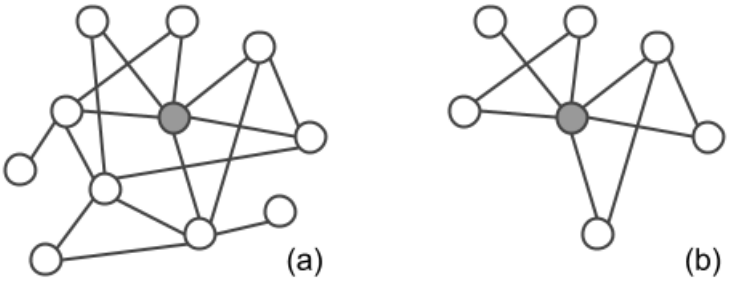

Figura 2.4: Ejemplo de red ego. (a) muestra la red completa y (b) la red ego asociada al nodo coloreado del grafo en (a). En (b) el nodo coloreado es el *ego* y el resto son los *alters*

Definición 2.1.15 *Se llama **comunidad** o **cluster** a cada una de las divisiones de una red en módulos. Los nodos de una misma comunidad suelen estar muy conectados entre sí.*

Para evaluar la calidad de las posibles comunidades se utilizan diferentes métricas. A continuación se van a detallar algunas.

2.1.1. Métricas de evaluación de comunidades

Tras la definición de algunos conceptos de teoría de grafos, se van a definir las métricas que se usarán para evaluar la calidad de las comunidades en las que un algoritmo ha dividido un grafo.

Definición 2.1.16 *La **conductancia** [16] de una comunidad S de un grafo $G = (V, E)$ mide la fracción total de enlaces que apuntan fuera de la comunidad.*

$$conductancia = \frac{\sum_{i \in S, j \notin S} a_{ij}}{\text{mín}(a(S), a(\bar{S}))}, \tag{2.1}$$

donde $a(S) = \sum_{i \in S} \sum_{j \in V} a_{ij}$, siendo a_{ij} los coeficientes de la matriz de adyacencia del grafo G.

Definición 2.1.17 *La **cohesividad** [17] caracteriza la estructura interna de una comunidad. Intuitivamente, una buena comunidad tiene que estar bien conectada internamente, tiene que ser relativamente difícil de dividir. Formalmente,*

$$cohesividad = \min_{S' \subset S} \phi(S'), \tag{2.2}$$

donde $\phi(S')$ es la conductancia de S' medido como subgrafo de S. Como se ha dicho, la conductancia mide el ratio de enlaces en S' que apuntan fuera del conjunto y los enlaces de dentro de S'. Una buena comunidad tiene una cohesividad alta.

Definición 2.1.18 *La **separabilidad** mide lo separados que está una comunidad del resto del grafo. Esta métrica mide el ratio entre el número de enlaces internos y externos en una comunidad*

$$separabilidad = \frac{|\{(u, v) \in E : u \in C, v \in C\}|}{|\{(u, v) \in E : u \in C, v \notin C\}|}, \tag{2.3}$$

2.2. Algoritmos de detección de comunidades

El objetivo de los algoritmos de detección de comunidades es agrupar la red en comunidades o *clusters*, similar al de la división de grafos (*graph partitioning*). Este problema puede resolverse con varios tipos de algoritmos [18], pero hay dos técnicas diferentes que se utilizan recurrentemente: maximizar la modularidad y usar *edge betweenness*.

Definición 2.2.1 *La **modularidad** es la diferencia entre el número de enlaces internos de una comunidad menos el número esperado en una red equivalente con enlaces colocados al azar [19]. Mide lo buena que es una comunidad, en el sentido de que hay muchos enlaces dentro de la comunidad y pocos entre comunidades [20].*

Considerando una partición del grafo $G = (E, V)$ en k particiones, $P = (P_1, P_2, ..., P_k)$, siendo m el número total de enlaces, la modularidad se calcula como:

$$Q = \frac{1}{2m} \left(\sum_{x=1} \sum_{v_i, v_j \in P_x} A_{ij} - \frac{d_i d_j}{2m} \right) \tag{2.4}$$

Definición 2.2.2 ***Edge betweenness** de un enlace e es el número de caminos más cortos, SP (shortest path), entre cada par de nodos de un grafo que pasan por ese enlace e [18].*

Antes de describir los algoritmos que se estudiarán en el trabajo, se va a explicar el método *Edge Betweenness Centrality*, EBC, de Girvan y Newman[21] ya que aparecerá en la descripción de algunos algoritmos y se basa en la utilización de *edge betweenness*, que se acaba de definir:

Algoritmo 1: Método Edge Betweenness Centrality

Input: G=(V,E)
while $E \neq \emptyset$**:**
    B = edgeBetweenness(E) `/* Edge betweenness de todos los enlaces en E */`
    e = highestEdgeBetweenness(E)
    $E = E \setminus \{e\}$ `/* Eliminar enlace con mayor edge betweenness */`

Las componentes conexas en cada iteración son las comunidades en las que se va dividiendo la red hasta llegar a que cada nodo es una comunidad. Mediante un criterio de parada se decide cuando parar de iterar para quedarse con la última división en comunidades que se haya hecho.

2.2.1. Algoritmos

A continuación se van a describir algunos de los algoritmos más comunes del problema de detección de comunidades.

- **Bigclam** (Cluster Affilitation Model for Big Networks) [1]

  Enfocado para comunidades con solapamientos, es decir, en las que un nodo puede pertenecer a más de una comunidad. Se basa en la observación de que los solapamientos entre comunidades tienden a ser más densos que las partes no solapadas, ya que entre más comunidades compartan dos nodos, más probable es que exista un enlace entre ellos. Por tanto, calcula la probabilidad de que un par de nodos estén conectados en función de que comparten comunidad.

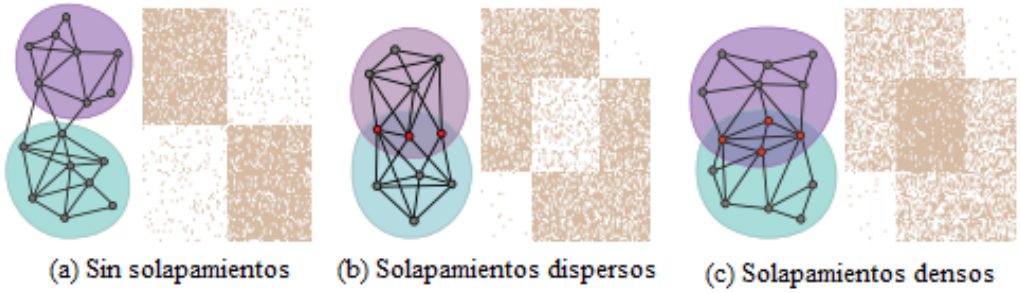

Figura 2.5: Estructura de redes. (a) Comunidades sin solapamiento. (b) Dos comunidades solapadas. (c) Solapamiento entre comunidades más denso. Izquierda: Red; Derecha: Matriz de adyacencia. Imagen obtenida de [1]

- **Fastgreedy** [22]

  Este algoritmo comienza con un estado en el cual cada vértice pertenece a una comunidad diferente. Entonces, se juntan en parejas para formar comunidades, eligiendo la unión que obtenga un mayor aumento de la modularidad. Este proceso puede representarse como un árbol cuyas hojas son los vértices de la red original y los nodos internos se corresponden a las uniones,de tal forma que queda un dendograma que representa la descomposición jerárquica de la red en comunidades en todos los niveles [20].

- **Clauset Newman Moore** [20]

  Este algoritmo jerárquico aglomerativo trata de optimizar la modularidad para definir comunidades siguiendo una estrategia *greedy*. La idea es la misma que Fastgreedy [22], pero, con una mejora de tiempo y memoria. Cuando se trabaja con grafos dispersos el *array* de enteros que representa la matriz de adyacencia tiene muchos valores nulos y hay un gasto de tiempo y memoria en el almacenamiento y mezcla de estos elementos. La mejora se consigue eliminando estas operaciones innecesarias.

- **Clique Percolation** [23]

  Este método busca comunidades usando k-cliques. Tras localizar los cliques de tamaño k, se forma un grafo en el cual los cliques se representan como nodos y dos nodos están unidos por un enlace si sus k-cliques comparten k - 1 nodos. Las componentes conexas de este grafo representan las comunidades buscadas. La imagen 2.6 muestra un ejemplo de esto.

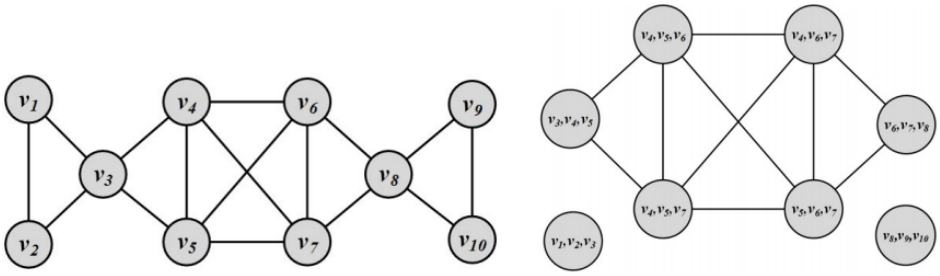

Figura 2.6: En (b) podemos ver el grafo resultante tras aplicar el método Clique Percolation con $k = 3$. Imagen obtenida de [2]

- **CoDA** (Communities through Directed Affiliations) [3]

  Este algoritmo es una mejora del Bigclam, anteriormente mencionado, detecta comunidades cohesivas (los nodos pueden enviar y recibir enlaces de otros miembros) y de 2-modos (cada nodo sólo envía o recibe enlaces) en redes dirigidas y no dirigidas.

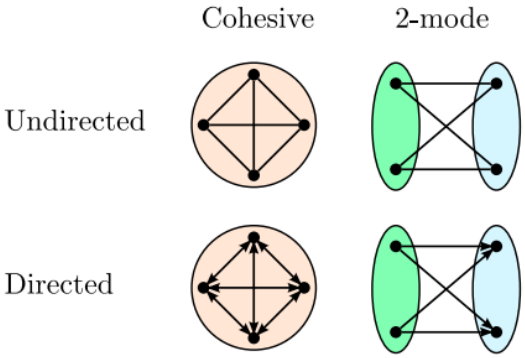

Figura 2.7: Tipos de redes (dirigidas y no dirigidas) y tipos de comunidades (cohesivas y 2-modo) que acepta el algoritmo CoDA. Imagen obtenida de [3]

- **CONGA** (Cluster-Overlapping Newman Girvan Algorithm) [4]

  Es una extensión del algoritmo EBC que permite a un nodo pertencer a varias comunidades. Para ello se introduce la división de vertices (*Splitting Vertices*), por el cual un vértice v, se divide en dos v_1 y v_2, de tal forma que puede pertenecer a varias comunidades.

Para decidir cuándo se debe dividir un vértice se introduce la noción *"split betweenness"*. Ésta consiste en contar el número de caminos más cortos, SP, que pasarían por un enlace entre v_1 y v_2. Si pasaran más SP por este enlace que por cualquier otro, el vértice se debe dividir, en otro caso, se elimina un vértice como en el EBC clásico.

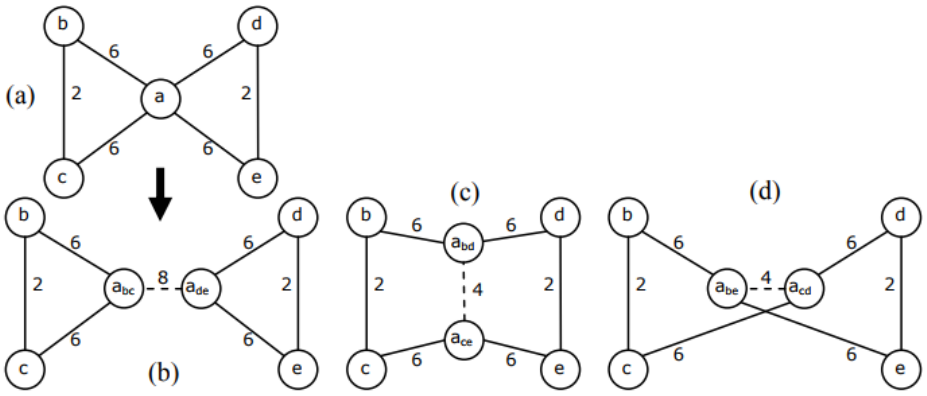

Figura 2.8: Ejemplo de *split betweenness*. (a) Red. (b) Mejor división del vértice a. (c), (d) Otras divisiones del vértice a. Imagen obtenida de [4]

- **CONGO** (CONGA Optimized Algorithm) [24]

  Para mejorar el alto coste computacional que hereda CONGA del EBC, aparece esta optimización. Calcular el *edge betweenness* es caro computacionalmente porque cuenta todos los SPs del grafo, por tanto, en esta versión solo se cuenta el SP localmente: en un pequeño subgrafo alrededor del enlace eliminado o el vértice dividido.

- **Infomaps** [5]

  Usa la probabilidad del flujo de paseos aleatorios (*random walks*) en una red. La red se descompone tratando de encontrar una partición que produzca la longitud mínima de descripción de un paseos aleatorio infinito en la red.

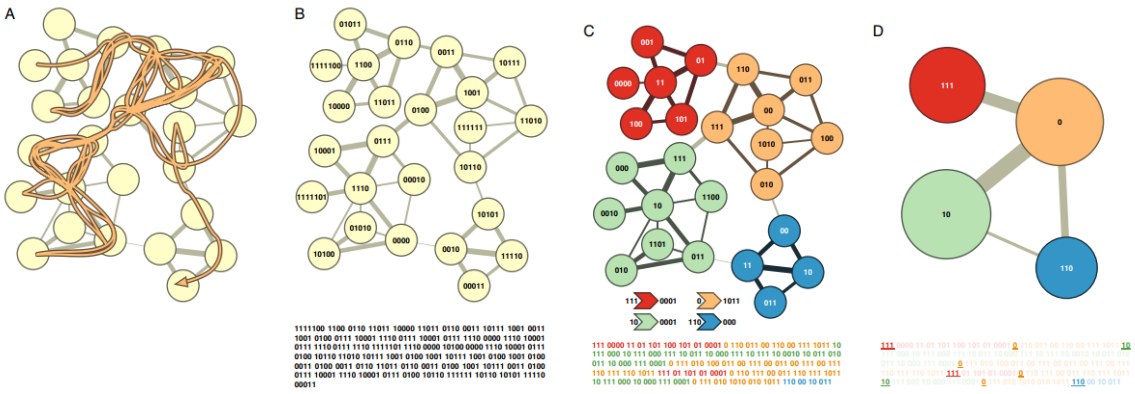

Figura 2.9: Ejemplo de ejecución del algoritmo infomaps. El tamaño de cada enlace indica la probabilidad de que un camino aleatorio pase por él. Imagen obtenida de [5]

- **Label Propagation** [6]

  Este algoritmo comienza suponiendo que cada nodo tiene una etiqueta que denota a qué comunidad pertenece. Entonces, cada nodo actualiza su etiqueta, eligiendo unirse a la comunidad a la que pertenezcan más vecinos. Así, las etiquetas se van propagando por la

red, de manera que los grupos más densos consiguen una única etiqueta. El algoritmo se detiene cuando todos los nodos tienen la etiqueta que la mayoría de sus vecinos.

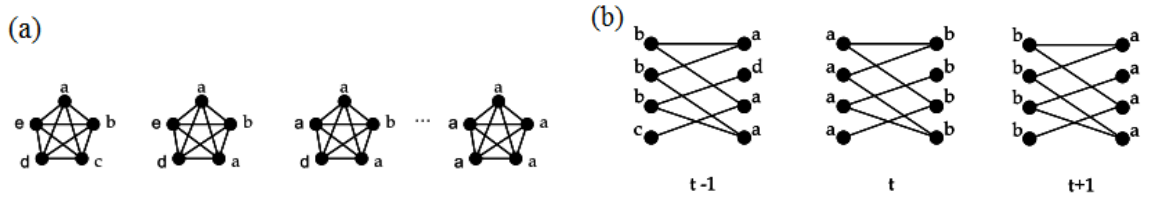

Figura 2.10: En (a) podemos ver como todos los nodos toman la misma etiqueta al aplicarle el *label propagation* debido a la alta densidad. (b) es un ejemplo en el que las etiquetas oscilan entre a y b debido a las elecciones en el paso t. Imagen obtenida de [6]

- **Leading Eigenvector** [25]

  Este algoritmo se basa en la optimización espectral de la modularidad mediante el uso de los autovalores (o valores propios) y autovectores (o vectores propios) de la matriz de modularidad. En primer lugar, se calcula el autovector líder de la matriz de modularidad, y luego el gráfico se divide en dos partes tratando de maximizar la mejora de la modularidad. Después, la contribución de la modularidad se calcula en cada paso de la subdivisión de una red. Se detiene una vez que el valor de la contribución de modularidad no sea positivo [26].

- **Multilevel** [27]

  En este algoritmo inicialmente todos los vértices pertenecen a una comunidad separada. Después, los vértices se mueven entre comunidades iterativamente con el objetivo de maximizar la modularidad. El algoritmo para cuando no es posible aumentar la modularidad.

- **Radicchi Strong** y **Radicchi Weak** [28]

  Este método funciona como el EBC pero cambia el criterio de eliminación de nodos (*edge betweenness*) para mejorar el coste computacional. En su lugar se utiliza el coeficiente de *clustering* del enlace, que se define a continuación:

  Definición 2.2.3 *El* ***coeficiente de clustering*** *o agrupamiento [29]* C_i *de un nodo* v_i *mide cómo de grupado está un nodo con sus vecinos. Esta medida para un vértice* v_i *está dada por la proporción entre los enlaces que conectan a sus vecinos dividido entre el número de enlaces existentes en un clique en el que la conectividad es máxima, es decir:*

$$C_i = \frac{|e_{jk}|}{k_i(k_i-1)}; e_{jk} \in E \tag{2.5}$$

  donde e_{jk} *son enlaces dirigidos (para grafos no dirigidos este valor se multiplica por 2) y* k_i *es el grado del nodo* v_i *del cual se está calculando el coeficiente.*

  Este algoritmo deja de eliminar enlaces cuando, al eliminar un enlace y dividirse el grafo, alguno de los subgrafos cumple una definición de comunidad previamente elegida:

  Definición 2.2.4 *El subgrafo* V *es una* ***comunidad en un sentido fuerte****, strong, si:*

$$k_i^{in}(V) > k_i^{out}(V), \forall i \in V. \tag{2.6}$$

  Y lo es ***en un sentido débil****, weak, si:*

$$\sum_{i \in V} k_i^{in}(V) > \sum_{i \in V} k_i^{out}(V). \tag{2.7}$$

- **Spinglass** [30]

  El principio básico del método es que los enlaces deben conectar nodos del mismo estado de giro (comunidad, en este contexto), mientras que los nodos de diferentes estados (que pertenecen a comunidades diferentes) no deben estar conectados. Por lo tanto, el objetivo de este algoritmo es encontrar el estado fundamental de un modelo de vidrio giratorio en el cual se minimice la energía libre del sistema [26].

- **Walktrap** [31]

  Este algoritmo se basa en la idea de que los paseos aleatorios cortos se mantienen dentro de una misma comunidad. Primero se calculan las distancias entre todos los nodos adyacentes mediante una nueva medida de similitud entre vértices. Luego, se eligen dos comunidades adyacentes, se fusionan en una nueva y se actualizan las distancias entre las comunidades. Este paso se repite (N - 1) veces.

2.3. Twitter

Primero se definirán algunos términos utilizados en Twitter que se utilizarán para determinar los diferentes modelos a estudiar y, después, se explicará cómo funciona la API con la cuál se han obtenido los datos que se han utilizado.

2.3.1. Términos de Twitter

A continuación se definirán varios términos relacionados con la red social de Twitter para aclarar a qué se refieren cuando aparezcan en el resto del documento.

Definición 2.3.1 *Un **tweet** es un mensaje de como máximo 280 caracteres (originalmente 140) que un usuario comparte en Twitter. Puede incluir fotos, videos o gifs.*

Definición 2.3.2 *El término **following** se refiere a los perfiles a los que un usuario **sigue**, es decir, los tweets que se publican por esas cuentas son los que aparecen en la página principal del usuario.*

Definición 2.3.3 ***Follower** o **seguidor** de un usuario es como se denota a los perfiles a los que les aparecen los tweets del usuario en cuestión.*

Definición 2.3.4 ***Reply** o **respuesta** es un tweet que contesta a otro tweet, de tal forma que aparecen vinculados. Solo se tendrán en cuenta cuando el usuario que publica el primer tweet y el que publica el segundo sean distintos.*

Definición 2.3.5 *Un **retweet** es reenviar un tweet. Esto quiere decir que si un usuario hace un retweet, el tweet aparecerá en las páginas principales de los followers del usuario que hizo el retweet.*

Definición 2.3.6 *Una **mención** o **quote** es un tweet en el cual se hace referencia a otro usuario mediante su identificador.*

2.3.2. API de Twitter

Twitter ofrece varias APIs (*Application Programming Interfaces*) para obtener datos de su red social [32], pero las más comunes para investigación son:

- *Streaming Tweets*

  Esta API permite la monitorización de Twitter a tiempo real. Existen dos opciones de usuario diferentes que ofrecen distintas cantidades y capacidades de filtros de *tweets*.

- *Search Tweets*

  Esta API de búsqueda permite búsquedas entre una muestra de *tweets* publicados en los últimos 7 días, 30 días o todo el archivo, dependiendo del tipo de usuario. Devuelve hasta 450 resultados cada 15 minutos en formato JSON.

Existen, a parte, bibliotecas de Python que facilitan el uso de la API de Twitter. Un ejemplo de una de ellas es Tweepy [33] o Twython [34], que ha sido la que se ha utilizado. Para hacer llamadas a la API de Twitter a través de estas librerías es necesario indentificarse mediante las claves de la aplicación, que tiene que haberse registrado previamente en la página de Twitter, y los *tokens* de usuario OAuth.

A la hora de hacer una búsqueda con la API *Search Tweets*, el parámetro necesario es una *query* de búsqueda y de manera opcional se puede añadir filtros como localización, lenguaje, tipo de *tweet* (popular o reciente), *tweets* por página o límites de fecha. La respuesta en formato JSON devuelve bastantes datos entre los que se incluyen: si el *tweet* se ha marcado como favorito o ha sido *retweeteado*, cuándo se creó, los *hashtags* y menciones a usuarios que incluye, a quién responde, el texto, el idioma y la información sobre el usuario que lo ha publicado. Entre la información del usuario se devuelve el nombre del usuario y su identificador, la imagen de perfil, cuándo creó la cuenta, su localización, el número de followers, su descripción...

Hay un ejemplo de respuesta JSON a una petición a la API de Twitter en el apéndice A.

3

Modelos de representación de las redes sociales

A la hora de representar una red social mediante un grafo, queda claro que los nodos representarán a los usuarios, pero, ¿cómo se representan los enlaces? La primera idea que se puede tener es representarlos teniendo en cuenta si son amigos (como sería en Facebook) o se siguen (como es el caso de Twitter). En el primero se obtendría un grafo no dirigido, mientras que el segundo sería dirigido, como se puede ver en la figura 3.1.

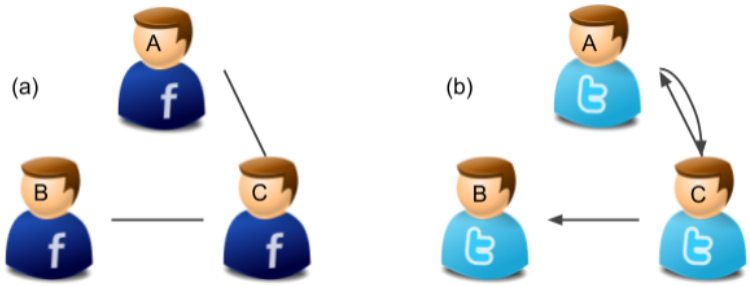

Figura 3.1: Ejemplo de representacion de las redes sociales (a) Facebook y (b) Twitter mediante grafos (a) no dirigido y (b) dirigido. En (a) A-C y B-C son amigos. En (b) C sigue a A y a B, mientras que A sigue a C

Tomando como base este modelo, se pueden hacer variaciones. Un ejemplo sería transformar el grafo que representa la red social de Twitter a no dirigido. Para ello basta con decir que existe un enlace entre u_1 y u_2 si u_1 sigue a u_2 o u_2 sigue a u_1. De esta forma, el grafo representado en la figura 3.1 (b) quedaría como la 3.1 (a).

Estos modelos, el de amistad de Facebook y el *following-follower* de Twitter, son los más comunes y utilizados en estudios de investigación [11, 12, 35, 36]. Como podemos ver, estos modelos ignoran las interacciones entre los usuarios. En particular, la red social que se va a estudiar en este trabajo es Twitter y en este escenario se producen muchas interacciones entre usuarios que no se siguen pero demuestran su afinidad (o lo contrario) respondiendose a *tweets* o *retweeteandose*. Con el modelo previamente descrito se pierde toda esta información.

Para solucionar ese problema, este trabajo pretende estudiar el resultado de un modelo alternativo, en el cual los enlaces representan las interacciones entre los usuarios. Por tanto, se entenderá que dos usuarios están conectados si:

- Un usuario menciona (*quote*) a otro usuario
- Un usuario responde (*reply*) a otro
- Un usuario hace *retweet* a otro

La figura 3.2 presenta las diferencias entre los dos modelos en un ejemplo de una posible situación real. En esta situación, un usuario A interactúa con otros dos: *retweetea* un *tweet* de B y responde un *tweet* de C, pero estas interacciones se pierden en la representación en forma de grafo con el modelo estándar, como muestra 3.2 (b). En cambio, la representación del grafo en 3.2 (c), con el nuevo modelo preserva todas las interacciones.

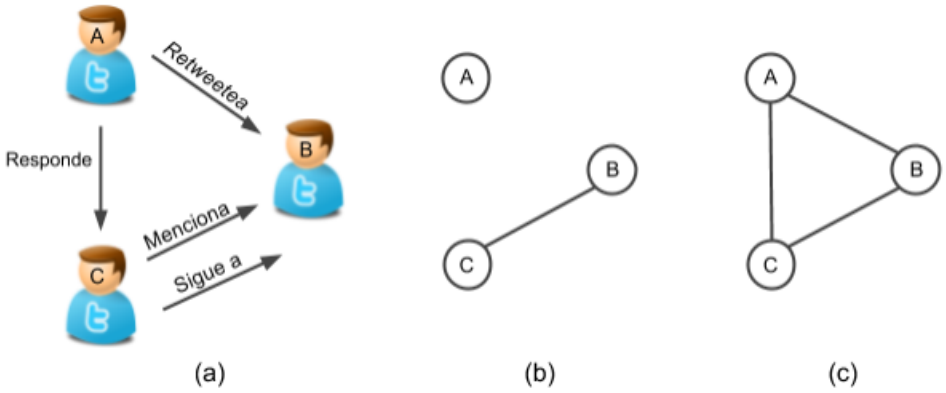

Figura 3.2: (a) Ejemplo de situacion real. (b) Grafo que representa la red (a) con el modelo *following-follower* estándar. (c) Grafo que representa la red (a) con el modelo propuesto

En el capítulo 5 se mostrarán los resultados de aplicar algoritmos de detección de comunidades en grafos generados con estos modelos. Además, también veremos los resultados que dan al tener en cuenta cada una de las interacciones por separado. La figura 3.3 muestras estas posibles respresentaciones.

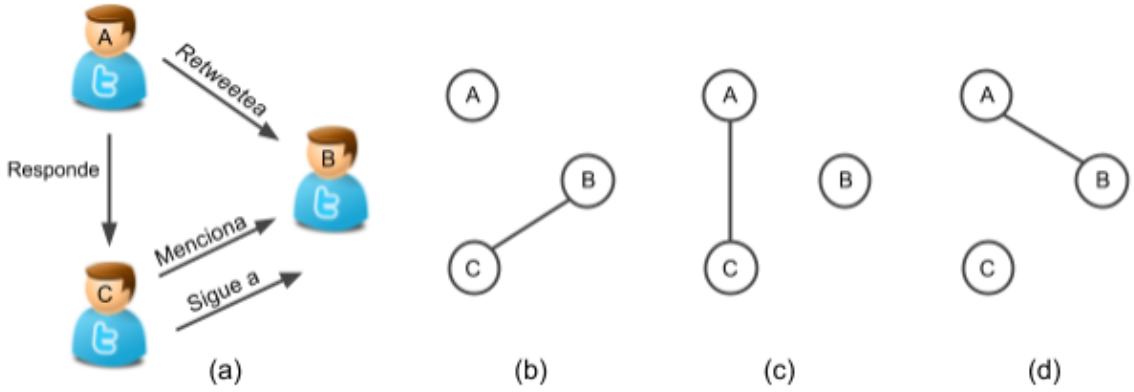

Figura 3.3: (a) Ejemplo de situacion real. Grafos que representa la red (a) teniendo en cuenta solo: (b) menciones, (c) respuestas y (d) retweets

4

Arquitectura

En este capítulo se describe la arquitectura desarrollada para el estudio de los algoritmos. Primero se comentan las herramientas que se han empleado para desarrollar el trabajo, después, el modelo de datos utilizado y, finalmente, se detalla en qué consiste la arquitectura.

4.1. Herramientas empleadas

Para desarrollar este trabajo, se han utilizado principalmente dos herramientas: una base de datos en Neo4j y Circulo. A continuación se describe cada una de ellas, comentando cómo se han utilizado.

4.1.1. Base de datos Neo4j

Las redes sociales son fácilmente modelables a partir de grafos, razón por la que se utilizará Neo4j como base de datos, ya que está orientada a grafos. Esta base de datos utiliza un lenguaje propio para hacer consultas a la base de datos: *Cypher*. Cypher es un lenguaje inspirado en SQL, para describir patrones en grafos visualmente usando una sintaxis de *ascii-art* [37]. A continuación se ve un ejemplo de una consulta que busca los usuarios que alguna vez han respondido al usuario con el nombre *twitter_nick*.

```
MATCH (u1:User) <-[:POSTED_BY]- (t1:Tweet) -[:REPLIES]- (t2:Tweet)
-[:POSTED_BY]-> (u2:User) where u2.screen_name=twitter_nick AND
u2.screen_name <> u1.screen_name RETURN DISTINCT u1.screen_name
```

En el modelo de datos se representan como nodos tanto a los usuarios como a los *tweets* de tal forma que los enlaces son las relaciones que existen entre ellos. En la sección 4.2 se describirá el modelo de datos con más detalle.

4.1.2. Circulo

Circulo [14] es un *framework* en *Python* que sirve para evaluar el resultado de algoritmos de detección de comunidades. Por defecto consta de 15 algoritmos implementados por *Lab41* [14], *igraph* [38] y SNAP [39]. Su ejecución se puede dividir en varias fases que se explican a continuación.

1. **Carga de datos**

   Antes de proceder a ejecutar los algoritmos y las métricas, es necesario preparar los datos para que tengan un formato consistente. Para ello, circulo dispone de un módulo, *data*, en el cual hay que crear un directorio por cada conjuntos de datos y se debe añadir un fichero *run.py* que se encargaría de descargar el conjunto de datos y tratarlo correctamente. En este caso, los datos se trataran mediante un *script* externo, convirtiéndolos a formato *graphml*, compatible con Circulo, por lo que en esta fase, simplemente se cargan los datos del directorio en el que se encuentran.

2. **Ejecución de algoritmos**

   La siguiente fase consistiría en ejecutar los algoritmos de detección de comunidades con los datos en el formato requerido, de lo cual se encarga *run_algos.py*. Para ejecutar los algoritmos se selecciona los datos y los algoritmos a ejecutar y el directorio para la salidad de los resultados de los respectivos algoritmos. En esta fase, primero se transforman los datos para hacerlos coincidir con los parámetros de entrada de cada algoritmo. Dado que los algoritmos pueden variar en la forma en que usan los enlaces ponderados, múltiples o dirigidos, es necesario transformar los datos para permitir que se ejecuten de manera adecuada. El comando con el que se ejecuta *run_algos.py* es el siguiente:

   ```
   python3 run_algos.py --output alg_output_dir dataset_name ALL
   ```

   En el lugar de *alg_output_dir* se pone el nombre del directorio en el cual se quiere guardar la salida de la ejecución, *dataset_name* representa el nombre del conjunto de datos con el que ejecutar los algoritmos y *ALL* representa que se quieren ejecutar todos lo algoritmos. En caso de querer ejecutar solo un algoritmo, basta poner en lugar de *ALL* el nombre del algoritmo: *bigclam, clauset_newman_moore, clique_percolation, coda, conga, congo, edge_betweenness, fastgreedy, infomap, label_propagation, leading_eigenvector, multilevel, radicchi_strong, radicchi_weak, spinglass, walktrap.*

3. **Ejecución de métricas**

   Por último se ejecutan las métricas, con las que se pueden sacar varias conclusiones:

   a) La efectividad del algoritmo comparando el las comunidades resultantes con las comunidades del *ground truth* (en este caso no se dispondrá de *ground truth*)
   b) La varianza de las comunidades resultantes, comparandolas con las producidas por otros algoritmos.
   c) La valoración de las comunidades resultantes en función de una serie de métricas
   d) La precisión de los algoritmos en función del tiempo

   Esta ejecución, de *run_metrics.py*, se hace mediante el comando:

   ```
   python3 run_metrics.py alg_output_dir metrics_output_dir dataset_name
   ```

   Donde *alg_output_dir* es el directorio donde se guardó la salida tras ejecutar *run_algos.py*, *metrics_output_dir* es el directorio donde se quiere guardar la salida de esta ejecución y *dataset_name* representa el nombre del conjunto de datos con el que ejecutaron los algoritmos.

4.2. Modelo de datos

Como se ha comentado anteriormente los usuarios y *tweets* se representan como nodos y los enlaces serán las relaciones que existan entre ellos, es decir:

- Un usuario **sigue a** (*follow*) otro usuario
- Un *tweet* es **publicado por** un usuario
- Un *tweet* **menciona** (*quote*) a otro usuario
- Un *tweet* **responde** (*reply*) a otro *tweet*
- Un *tweet* hace ***retweet*** otro *tweet*

En la base de datos, estas posibles relaciones quedarían como se muestran en la figura 4.1, teniendo en cuenta que los usuarios y los *tweets* serían nodos.

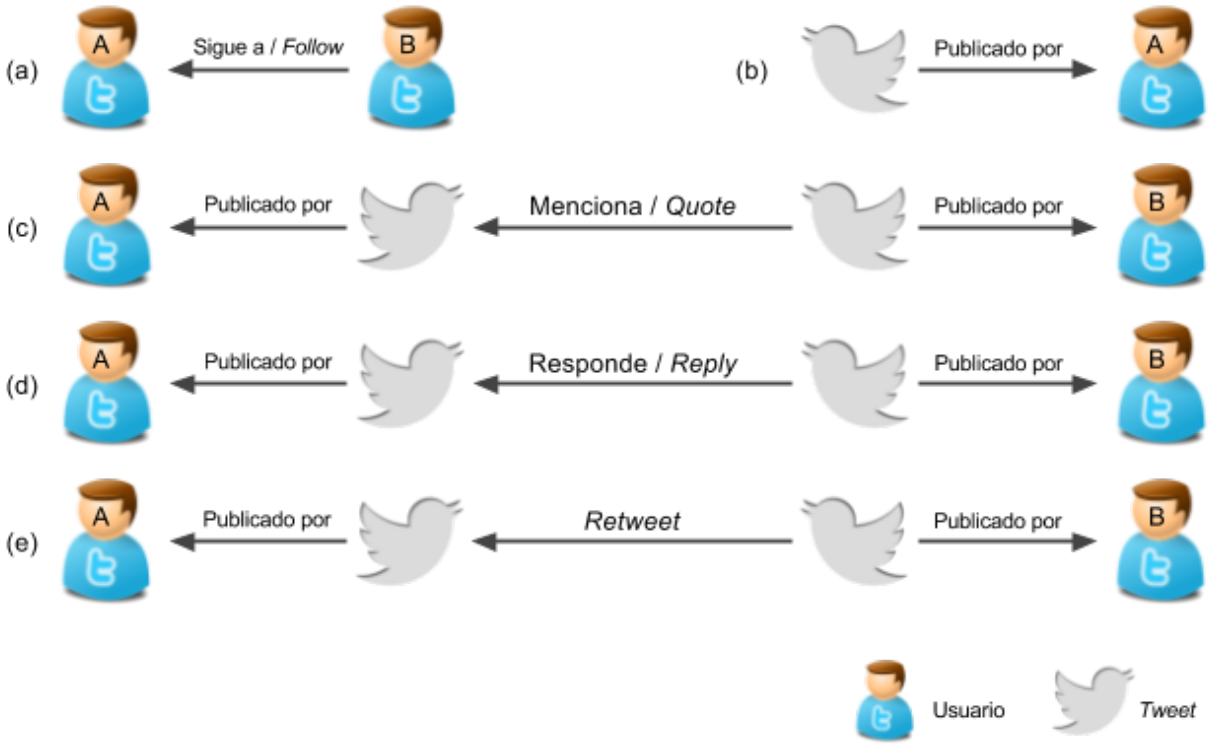

Figura 4.1: Representación en la base de datos de lo que comúnmente sería: (a) B sigue a A (b) A publica un *tweet* (c) B menciona a A (d) B responde a A (e) B *retweetea* un *tweet* de A

4.3. Sistema

Tras ver las herramientas que se han utilizado y el modelo de datos, se va a explicar en qué consiste el sistema que se ha construído.

La figura 4.2 muestra todo el proceso que es necesario seguir para obtener los resultados finales. Los datos se extraen de Twitter y se guardan en la base de datos Neo4j. Después se obtienen las redes ego y se guardan en el formato compatible con Circulo, mediante un

Figura 4.2: Representación del sistema

preprocesado. Luego se ejecuta Circulo y, finalmente, se obtienen los resultado que se quieres de la salida de Circulo.

A continuación se describe con más detalle el sistema y las modificaciones de Circulo que han sido necesarias para el correcto estudio de este modelo de datos.

4.3.1. Preprocesado

Para extraer los datos de la base de datos en Neo4j en el formato deseado se ha creado un *script* con ayuda de la librería *Py2neo* [40]. En este *script* se generan las redes ego a analizar. A continuación puede verse un fragmento de pseudocódigo del algoritmo que se utiliza para generar las redes ego.

Algoritmo 2: Obtención de las redes ego según un criterio de vecindad dado

```
Input: criterio_vecindad
users = get_all_users()
for u ∈ users:
    g = Graph()
    alters = get_alters(u, criterio_vecindad)        /* Devuelve los vecinos de u */
    for alter ∈ alters:
        g.add_edge(u, alter)                    /* Enlace entre el ego y su alter */
        alt_alter = get_alters(alter, criterio_vecindad)
        common_alters = alters ∩ alt_alters
        for common ∈ common_alters:
            g.add_edge(alter, common)             /* Enlace entre los alters */
    file.write(g)                                    /* Crea fichero graphml */
```

Cabe hacer notar que criterio de vecindad se refiere a la relación que se usa para decir que dos nodos son vecinos. En este trabajo se utilizarán los siguientes criterios de vecindad:

- **Respuesta**: dos nodos usuarios son vecinos sólo si uno ha respondido algún *tweet* publicado por el otro.
- ***Retweet***: dos nodos usuarios son vecinos sólo si uno ha *retweeteado* un *tweet* publicado por el otro.
- **Mención**: dos nodos usuarios son vecinos sólo si uno ha publicado un *tweet* mencionando al otro.
- **Todo lo anterior**: dos nodos usuarios son vecinos si uno ha respondido o *retweeteado* un *tweet* publicado por el otro o ha mencionado al otro.
- ***Follow***: dos nodos usuarios son vecinos sólo si uno sigue o hace *follow* a otro.

De esta manera, se han generado cinco conjuntos de datos formados por redes ego, uno por cada uno de los criterios de vecindad definidos arriba.

El *script* de preprocesado se asegura que todas las comunidades tengan el formato que espera Circulo, *graphml*. Para ello se utiliza igraph [38], una librería para trabajar fácilmente con grafos.

4.3.2. Ejecución

La siguiente fase consiste en la ejecución mediante Circulo de todos los algoritmos sobre todos los conjuntos de datos, para lo cual existen los ficheros: *run_algos.py* y *run_metrics.py*. Puesto que se ha querido estudiar la red a través de sus redes ego, ha sido necesario modificar estos ficheros, ya que estaban preparados para coger los datos de entrada de un solo fichero. Al haber dividido la red en sus redes ego, los datos de entrada estaban divididos en varios ficheros, uno por cada miembro de la red, cada uno con su red ego. Por tanto, el código se ha modificado para que los datos se cogieran de un directorio, recorriendo todos los ficheros. La figura 4.3 representa la modificación que se ha hecho.

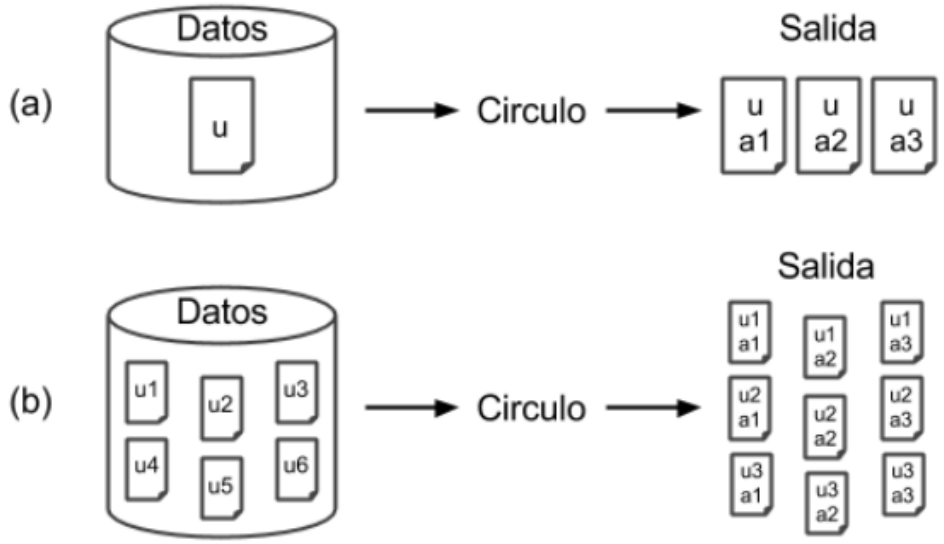

Figura 4.3: Modificación de circulo. (a) representa cómo es la entrada de datos y la salida tras ejecutarlo. (b) muestra cómo queda tras la modificación realizada. Cada fichero u o u_i representa una red ego y, en la salida, a_i se refiere al resultado de cada algoritmo

La salida de *run_algos.py* consiste, tras la modificación, en un directorio por cada conjunto de datos con un fichero por cada usuario (u_i) y algoritmo (a_i) ejecutado, con los respectivos resultados. Estos resultados se evalúan mediante *run_metrics.py* resultando, de la misma manera en un fichero JSON por cada usuario y algoritmo evaluado.

4.3.3. Estudio de resultados

Como se ha dicho en el apartado anterior, la salida final de Circulo es un directorio con un fichero JSON por cada usuario y algoritmo evaluado. Puesto que se busca evaluar cada algoritmo sobre cada uno de los conjuntos de datos (aunque estos estén divididos en redes ego), tenemos que agrupar los resultados de las redes ego que han sido ejecutadas para un mismo algoritmo. Por otro lado, Circulo proporciona mucha información en la salida de *run_metrics.py* pero,

en este trabajo, se van a utilizar tres métricas para el análisis de los algoritmos: cohesividad, conductancia y separabilidad.

Por lo tanto, para el estudio de resultados se ha desarrollado un *script* que se encarga de filtrar las métricas deseadas y agruparlas, tomando las medias, cuando sean el resultado de un mismo algoritmo. Finalmente, obtenemos un fichero por cada conjunto de datos y cada uno contiene la media y la varianza de las tres métricas para cada uno de los algoritmos.

5

Experimentos realizados y resultados

A continuación se van a exponer los resultados que se han obtenido tras la experimentación llevada a cabo sobre la arquitectura propuesta en la sección anterior. Primero se detallarán las características de cada conjunto y, después, se analizará el resultado de los algoritmos con cada uno de los conjuntos de datos. Esto se repetirá aplicando un filtro mayor y, por último, se mostrará un ejemplo gráfico de redes.

5.1. Datos

Se van a probar los algoritmos en diferentes modelos de datos, según sus interacciones en Twitter, partiendo de un total de 12519 usuarios. Debido a que los datos utilizados no están contrastados y al nuevo modelo propuesto, pueden surgir un gran número de redes ego muy pequeñas porque el *ego* no realice alguna de las interacciones que se van a tener en cuenta, o lo haga con poca frecuencia. Por esto, se han filtrado las redes ego que tienen 3 o más nodos, ya que las menores de ese tamaño son demasiado pequeñas como para dividirlas en comunidades y no proporcionan ninguna información. Tras este filtrado, las características de las redes analizadas se muestran en las tablas 5.1 y 5.2. Ambas muestran el número mínimo, máximo y de media que tienen de nodos y de enlaces, respectivamente, la varianza de la media y el número de redes ego que tiene cada uno de los conjuntos de datos.

Datos	**Mínimo**	**Máximo**	**Media**	**Varianza**	**Egos**
Menciones	3	134	4	103.1	367
Respuestas	3	479	7	737.4	801
Retweets	3	921	16	4586.5	862
MRR	3	925	9	2258.3	2340
Follows	8	401	149	12875.4	7

Cuadro 5.1: Número de nodos mínimo, máximo y de media, varianza de la media y número de redes ego que componen cada conjunto de datos que se han analizado. MRR incluye Menciones, Respuestas y *Retweets*

Datos	**Mínimo**	**Máximo**	**Media**	**Varianza**	**Egos**
Menciones	2	192	4	175.6	367
Respuestas	2	514	7	826.6	801
Retweets	2	920	16	4693.6	862
MRR	2	1116	9	2719.3	2340
Follows	25	2704	1024	669873.4	7

Cuadro 5.2: Número de enlaces mínimo, máximo y de media, varianza de la media y número de redes ego que componen cada conjunto de datos que se han analizado. MRR incluye Menciones, Respuestas y *Retweets*

En las tablas, 5.1 y 5.2, se puede ver que el conjunto de datos formado por las menciones tiene un número medio de nodos y de enlaces muy bajo, pero también es el que se compone por menos redes ego (sin contar la red compuesta por los *follows*). Esto puede conllevar resultados extraños a la hora de formar comunidades, ya que, tomando los valores medios de nodos y enlaces, muchas de las redes que se van a estudiar serían como se muestra en la figura 5.1.

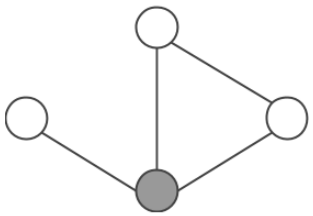

Figura 5.1: Ejemplo de red ego formada por cuatro nodos y cuatro enlaces. El nodo *ego* es el coloreado

En cuanto al resto de conjunto de datos, obtenemos unos valores medios aceptables. Y, en cuanto a la relación entre nodos y enlaces: en [18] se dice que la identificación de *clusters* es solo posible en grafos dispersos ya que si el número de enlaces es mucho mayor que el de nodos, la distribución es demasiado homogénea como para que una división en comunidades tenga sentido. Teniendo en cuenta esto, podemos decir que las redes obtenidas son aptas para el análisis de comunidades. El conjunto de datos formado por *Follows* solo consta de las redes ego de 7 usuarios porque obtener los datos supone un alto coste computacional.

5.2. Resultados del experimento

Para comparar los diferentes modelos propuestos, se han ejecutado los algoritmos descritos en la sección 2.2, con ayuda del *framework* Circulo. En las siguientes tablas se va a ver el resultado final obtenido tras la ejecución del *script* de procesado de la salida de Circulo. En ellas se muestra el resultado obtenido de las métricas (conductancia, cohesividad y separabilidad) para cada uno de los modelos. Se busca una baja conductancia, alta cohesividad y baja separabilidad. En cada tabla de marcaran en negrita los mejores valores y en cursiva valores *demasiado* buenos.

Para verificar que los mejores valores eran debido a una buena división de la red en comunidades y no a una división trivial, se ha obtenido el número de comunidades en las que se ha dividido cada red, y cuántas redes egos se han dividido en ese número de comunidades. Estos datos se representan en forma de diccionario en una tabla por cada conjunto de datos. Las tablas pueden consultarse en el apéndice B.1.

5.2.1. Menciones

En la tabla 5.3 se obtienen valores perfectos en Bigclam, Clique Percolation y Radicchi Strong. A parte, se han marcado en negrita los siguientes mejores valores.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Bigclam	*0.0000*	0.0000	**0.9922**	0.0025	*0.0000*	0.0000
Clauset Newman Moore	0.0445	0.0191	0.9808	0.0070	0.1139	0.3113
Clique Percolation	*0.0000*	0.0000	0.9681	0.0111	*0.0000*	0.0000
Coda	0.0575	0.0325	0.9425	0.0260	0.1945	0.3848
Conga	**0.0005**	0.0000	0.8847	0.0352	**0.0771**	0.3792
Congo	**0.0005**	0.0000	0.8918	0.0340	0.0787	0.3838
Fastgreedy	0.0233	0.0088	0.9630	0.0151	0.0898	0.2638
Infomap	0.0025	0.0011	0.9403	0.0257	**0.0037**	0.0025
Label Propagation	0.0155	0.0096	0.9383	0.0263	0.0422	0.3640
Leading Eigenvector	0.0194	0.0068	0.9614	0.0155	0.1372	1.1428
Multilevel	0.0456	0.0187	0.9869	0.0045	0.1117	0.2710
Radicchi Strong	*0.0000*	0.0000	0.9385	0.0267	*0.0000*	0.0000
Radicchi Weak	0.0092	0.0061	0.9482	0.0220	0.0181	0.0252
Spinglass	0.0804	0.0372	**0.9887**	0.0039	0.1756	0.3996
Walktrap	0.0192	0.0070	0.9624	0.0155	0.0936	0.2913

Cuadro 5.3: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Menciones. Mejores resultados en negrita

En los casos de Bigclam, Clique Percolation y Radicchi Strong que obtenían valores óptimos de conductancia y separabilidad, efectivamente se puede ver, observando la tabla B.1, que son los que menos divisiones en comunidades han hecho, asignando a la mayoría de nodos la misma comunidad (o ninguna, en el caso de Bigclam). Esa es la razón de sus resultados óptimos en cuanto a conductancia y separabilidad.

Por tanto, teniendo en cuenta los resultados de 5.3 y viendo las divisiones que han hecho estos algoritmos en la tabla B.1, se pueden tomar CONGA, CONGO y Spinglass como los algoritmos que obtienen los mejores resultados, ya que han hecho una división en *clusters* mayor. Infomap solo divide en comunidades dos redes ego, por lo que tampoco la tenemos en cuenta. Por tanto, se tiene que CONGA y CONGO obtienen mejor conductividad y Spinglass mejor cohesividad. En cuanto a algoritmos candidatos para mejor separabilidad, a la vista de los resultados, nos quedamos con CONGA que obtiene el siguiente mejor resultado en la tabla 5.3.

5.2.2. Respuestas

En la tabla 5.4, se ve que Bigclam y Clique Percolation obtienen valores perfectos y los siguientes mejores resultados son de Clauset Newman Moore, CONGA, CONGO, Multilevel y Radicchi Strong.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Bigclam	*0.0000*	0.0000	**0.9967**	0.0011	*0.0000*	0.0000
Clauset Newman Moore	0.0420	0.0160	**0.9920**	0.0029	0.2579	1.9496
Clique Percolation	*0.0000*	0.0000	0.9796	0.0082	*0.0000*	0.0000
Coda	0.0918	0.0665	0.9198	0.0499	0.1092	0.2052
Conga	**0.0003**	0.0000	0.9037	0.0307	**0.0168**	0.0788
Congo	**0.0003**	0.0000	0.9043	0.0305	**0.0169**	0.0788
Fastgreedy	0.0306	0.0109	0.9764	0.0100	0.2414	1.8170
Infomap	0.0062	0.0027	0.9407	0.0256	0.0666	0.5421
Label Propagation	0.0152	0.0087	0.9385	0.0260	0.4848	27.7804
Leading Eigenvector	0.0236	0.0081	0.9677	0.0139	0.3555	6.1292
Multilevel	0.0455	0.0172	**0.9919**	0.0029	0.2568	1.8175
Radicchi Strong	**0.0009**	0.0007	0.9354	0.0285	**0.0002**	0.0000
Radicchi Weak	0.0161	0.0110	0.9482	0.0219	0.0483	0.1759
Spinglass	0.0880	0.0416	0.9897	0.0037	0.2917	0.9477
Walktrap	0.0298	0.0108	0.9745	0.0109	0.2396	1.8167

Cuadro 5.4: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Respuestas. Mejores resultados en negrita

Observando la tabla B.2 se puede ver que Bigclam, Clauset Newman Moore, Clique Percolation y Radicchi Strong dividen la mayoría de redes ego en una o ninguna comunidades.

Por tanto, se tiene que CONGA, CONGO y Multilevel son, de entre los algoritmos que obtienen mejores valoraciones en cuanto a métricas, los que más divisiones hacen. CONGA y CONGO obtienen los mejores resultados de conductancia y separabilidad y Multilevel de cohesividad.

5.2.3. Retweets

En la tabla 5.5, se puede ver que se obtienen valores perfectos en Bigclam y Clique Percolation y los siguientes mejores resultados son de Clauset Newman Moore, CONGO y Radicchi Strong.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Bigclam	*0.0000*	0.0000	*1.0000*	0.0000	*0.0000*	0.0000
Clauset Newman Moore	0.0348	0.0112	**0.9932**	0.0028	0.9063	57.5226
Clique Percolation	*0.0000*	0.0000	*1.0000*	0.0000	*0.0000*	0.0000
Coda	0.1785	0.1212	0.8493	0.0885	0.8935	165.3372
Conga	0.0013	0.0000	0.8975	0.0260	0.1842	0.6515
Congo	**0.0007**	0.0000	0.8977	0.0248	**0.0912**	0.2745
Fastgreedy	0.0262	0.0079	0.9886	0.0050	0.8533	57.0588
Infomap	0.0071	0.0020	0.9603	0.0198	0.6494	55.7155
Label Propagation	0.0106	0.0060	0.9512	0.0247	0.0725	0.6855
Leading Eigenvector	0.0239	0.0071	0.9859	0.0058	0.8172	56.5332
Multilevel	0.0314	0.0102	0.9935	0.0026	0.8584	57.0525
Radicchi Strong	**0.0005**	0.0001	0.9465	0.0270	**0.0046**	0.0093
Radicchi Weak	0.0168	0.0062	0.9731	0.0118	0.2580	3.3215
Spinglass	0.0996	0.0543	0.9903	0.0038	0.4577	2.0390
Walktrap	0.0222	0.0069	0.9829	0.0078	0.8454	57.0663

Cuadro 5.5: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Retweets. Mejores resultados en negrita

De nuevo, mirando la B.3 se observa que los únicos algoritmos que dividen la red ego en un número razonable de comunidades son Clauset Newman Moore y, sobre todo, CONGO. Clauset Newman Moore consigue ser el mejor en cohesividad , mientras que CONGO, lo hace en conductancia y separabilidad.

El resto de algoritmos que obtenían mejores resultados apenas hacen divisiones en comunidades, por lo que se han descartado.

5.2.4. Menciones, respuestas y retweets

En la tabla 5.6, se puede ver que los mejores resultados son de Bigclam, Clique Percolation, CONGO, Radicchi Strong y Spinglass.

Algorithm	Media conduc.	Varianza conduc.	Media cohesiv.	Varianza cohesiv.	Media separa.	Varianza separa.
Bigclam	**0.0002**	0.0000	**0.9921**	0.0026	0.0035	0.0106
Clauset Newman Moore	0.0556	0.0212	0.9771	0.0078	0.2057	1.1705
Clique Percolation	0.0029	0.0021	0.9660	0.0124	**0.0010**	0.0002
Coda	0.0824	0.0514	0.9192	0.0403	0.1749	0.6713
Conga	0.0017	0.0007	0.9031	0.0311	0.0541	0.2202
Congo	**0.0013**	0.0006	0.9027	0.0312	0.0497	0.2287
Fastgreedy	0.0384	0.0139	0.9570	0.0166	0.2178	2.2258
Infomap	0.0087	0.0035	0.9205	0.0323	0.0945	1.5540
Label Propagation	0.0158	0.0092	0.9163	0.0347	0.3605	55.4617
Leading Eigenvector	0.0330	0.0117	0.9504	0.0201	0.3323	8.8973
Multilevel	0.0615	0.0232	0.9794	0.0070	0.2414	2.2284
Radicchi Strong	**0.0007**	0.0003	0.9103	0.0377	**0.0033**	0.0067
Radicchi Weak	0.0250	0.0164	0.9333	0.0263	0.0927	1.3271
Spinglass	0.0906	0.0398	**0.9801**	0.0072	0.2390	0.6538
Walktrap	0.0375	0.0149	0.9547	0.0182	0.2117	2.2242

Cuadro 5.6: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Menciones, Respuestas y *Retweets*. Mejores resultados en negrita

En la tabla B.4 se observa que Bigclam, Clique Percolation y, sobre todo, Radicchi Strong no realizan muchas divisiones en comunidades. Por tanto, se tiene que CONGO es el mejor en conductancia y Spinglass en cohesividad. En cuanto a separabilidad, observando de nuevo la tabla 5.6, se puede ver que CONGO obtiene el siguiente mejor valor.

5.2.5. Follows

En la tabla 5.7, se observa que los mejores resultados son de CONGO y Label Propagation y se obtienen valores perfectos con Radicchi Strong.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Clauset Newman Moore	0.4842	0.0081	0.4693	0.0107	0.7572	0.1815
Coda	0.3649	0.0324	0.6519	0.0021	1.5999	0.9058
Conga	0.0282	0.0020	0.6308	0.0069	41.4244	6287.4502
Congo	**0.0091**	0.0005	**0.7409**	0.0062	0.9416	5.3201
Fastgreedy	0.4878	0.0101	0.4255	0.0113	0.7135	0.1346
Infomap	0.1584	0.0357	0.5242	0.0352	5.8893	175.5491
Label Propagation	0.0246	0.0036	0.3951	0.0246	**0.3933**	0.9280
Leading Eigenvector	0.4373	0.0161	0.4189	0.0201	0.8903	0.4718
Multilevel	0.4555	0.0176	0.3870	0.0057	0.8321	0.5009
Radicchi Strong	*0.0000*	0.0000	0.3732	0.0282	*0.0000*	0.0000
Radicchi Weak	0.4945	0.1303	0.6851	0.0437	0.7687	0.7898
Spinglass	0.5231	0.0045	0.4900	0.0479	0.6101	0.0649
Walktrap	0.4917	0.0868	0.6257	0.0402	0.6695	0.4481

Cuadro 5.7: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de *Follows*. Mejores resultados en negrita

En la tabla B.6 se puede ver que Radicchi Strong asigna una sola comunidad a todas las ego, por lo que este algoritmo queda rechazado como candidato. Lo mismo pasa con Label Propagation que divide 6 de las 7 redes en una sola comunidad por lo que tambiém se descarta. En cambio, CONGO, sí hace divisiones, por lo que se puede tomar como mejor candidato a conductancia y cohesividad.

Mirando de nuevo 5.7, se ve que el algoritmo que obtiene el siguiente mejor resultado en cuanto a separabilidad es Spinglass.

5.2.6. Conclusiones

Como se ha visto, los algoritmos que menos divisiones han hecho han sido Bigclam, Clique Percolation, Label Propagation y Radicchi Strong. En el caso de Bigclam puede deberse a que, al tratarse de comunidades pequeñas en su mayoría y redes muy dispersas, no se encuentren zonas densas a tener en cuenta. Por la misma razón, puede ser que no se encuentren k-cliques suficientes para dividir la red en comunidades en Clique Percolation, y para el cálculo del coeficiente de *clustering* en Radicchi Strong. En el caso de Label Propagation, la propagación de una misma etiqueta por toda la red puede deberse también al tamaño reducido de las comunidades o la estructura que tenga la red.

A continuación, en la figura 5.2 se muestran tres gráficas en las que se va a poder comparar el resultado de las métricas en cada uno de los conjuntos de datos: Menciones (Men), Respuestas (Res), Retweets (Ret), Menciones, Respuestas y Retweets (MRR) Follows (Fol).

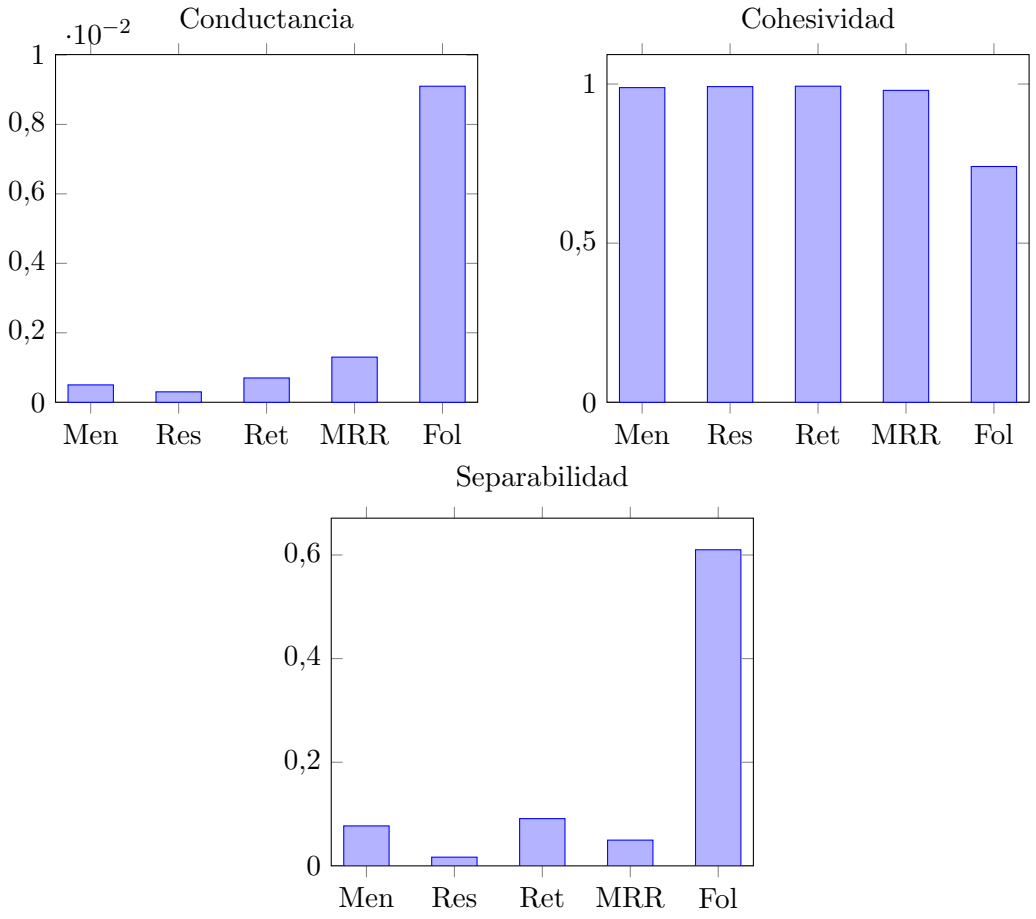

Figura 5.2: Gráficas que muestran los mejores resultados que han obtenido los cinco conjuntos de datos en cada una de las métricas. Arriba izquierda: conductancia, arriba derecha: cohesividad y abajo, separabilidad. Conjuntos de datos: Menciones (Men), Respuestas (Res), Retweets (Ret), Menciones, Respuestas y Retweets (MRR) Follows (Fol)

Observando las gráficas, se ve que el conjunto de datos que obtiene mejor conductancia es el de Respuestas (con CONGA y CONGO). En cuanto a cohesividad, es el conjunto de datos de Retweets (con Clauset Newman Moore) el que obtiene resultados mejores, aunque no dista mucho de Menciones (con Spinglass), Respuestas (con Multilevel) y el de los tres (con Spinglass también). Por último, el conjunto de datos formado por Respuestas (con CONGA y CONGO) es el que obtiene mejor separabilidad.

Por tanto, se puede concluir que el algoritmo que mejores resultados ha obtenido es CONGO, seguido de CONGA y Spinglass. Y el modelo de datos que ha resultado obtener mejores comunidades es el conjuntos de Respuestas.

Ante estos resultados, cabe hacer notar que el conjunto de datos utilizado para Menciones, Respuestas y *Retweets* está compuesto por más perfiles de Twitter que el utilizado en el conjunto compuesto por los *Follows*. Este mayor número de redes incluye, además, redes más pequeñas que se han podido dividir peor en comunidades y han afectado al resultado de las métricas. Esto se ha podido traducir en una mejor cohesividad o conductancia.

Esto nos lleva a plantearnos rehacer una ejecución con un conjunto de datos que tengan un número medio de nodos y enlaces más parecido.

5.3. Experimento con un filtro mayor

Se ha vuelto a realizar todo el proceso con el fin de obtener una mejor de idea de como funcionan los modelos alternativos frente al *following-follower*. Esta vez se han filtrado las redes ego formadas por más de 8 nodos, que es el número de nodos mínimo que tiene una red del conjunto de datos de *Follows*. Las características de los nuevas redes a analizar se muestran en las tablas 5.8 y 5.9.

Datos	**Mínimo**	**Máximo**	**Media**	**Varianza**	**Egos**
Menciones	8	134	25	1120.02	24
Respuestas	8	479	50	6040.68	73
Retweets	8	921	82	21954.68	146
MRR	8	925	74	21406.55	203
Follows	8	401	149	12875.4	7

Cuadro 5.8: Número de nodos mínimo, máximo y de media, varianza de la media y número de redes ego que componen cada conjunto de datos que se han analizado. MRR incluye Menciones, Respuestas y *Retweets*

Datos	**Mínimo**	**Máximo**	**Media**	**Varianza**	**Egos**
Menciones	7	192	31	1916.94	24
Respuestas	7	514	52	6829.1	73
Retweets	7	920	83	22251.28	146
MRR	7	1116	81	25694.29	203
Follows	25	2704	1024	669873.4	7

Cuadro 5.9: Número de enlaces mínimo, máximo y de media, varianza de la media y número de redes ego que componen cada conjunto de datos que se han analizado. MRR incluye Menciones, Respuestas y *Retweets*

En la tabla 5.8 se ve que las redes ego formadas por Menciones, Respuestas y Retweets siguen siendo más pequeñas que las formadas por *Follows*, pero, exceptuando el conjunto de las Menciones, tienen un número máximo de nodos mayor y ya se parecen más. Aunque la media de enlaces en las redes ego de *Follows* es bastante mayor, hay que hacer notar la varianza tan alta que tiene. Observando el valor mínimo de enlaces y nodos, se ve que va a haber comunidades con forma de estrella (figura 5.3) y, mirando la media, puede verse que en general las redes no van a ser muy densas.

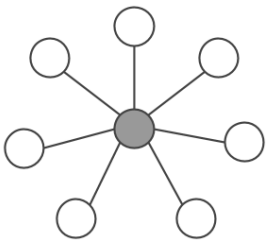

Figura 5.3: Ejemplo de red ego estrellada

De nuevo se han contado el número de redes ego que se dividen en el mismo número comunidades. Las tablas con esta información pueden consultarse en el apéndice B.2

5.3.1. Menciones

En la tabla 5.10 se obtienen valores perfectos en Clique Percolation y Radicchi Strong. A parte, se han marcado en negrita los siguientes mejores valores para cada métrica: CONGA, Infomap y Multilevel.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Clauset Newman Moore	0.1279	0.0319	0.8955	0.0251	0.8438	3.3188
Clique Percolation	*0.0000*	0.0000	0.8333	0.0278	*0.0000*	0.0000
Coda	0.4954	0.1525	0.5509	0.0807	0.3963	0.3696
Conga	**0.0044**	0.0001	0.8704	0.0081	0.6842	2.9498
Congo	0.0047	0.0001	0.8523	0.0058	0.6984	2.9733
Fastgreedy	0.2400	0.0386	0.9427	0.0144	1.1159	2.6754
Infomap	0.0380	0.0159	0.7314	0.0673	**0.0563**	0.0352
Label Propagation	0.0257	0.0073	0.7304	0.0699	0.1719	0.3982
Leading Eigenvector	0.1806	0.0308	0.9201	0.0190	1.8353	14.1092
Multilevel	0.2488	0.0432	**0.9739**	0.0040	1.1185	2.6802
Radicchi Strong	*0.0000*	0.0000	0.6826	0.0698	*0.0000*	0.0000
Radicchi Weak	0.1233	0.0715	0.7912	0.0569	0.2492	0.3100
Spinglass	0.3655	0.0453	0.9690	0.0044	1.6862	3.0315
Walktrap	0.1958	0.0353	0.9433	0.0128	1.2019	2.9251

Cuadro 5.10: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Menciones. Mejores resultados en negrita

En los casos de Clique Percolation y Radicchi Strong, que han obtenido valores óptimos de conductancia y separabilidad, efectivamente se puede ver, observando la tabla B.7, que son los que menos divisiones en comunidades han hecho, asignando a la mayoría de nodos la misma comunidad. Esa es la razón de sus resultados óptimos en cuanto a conductancia y separabilidad.

Por otro lado, Infomap, que era el que obtenía el siguiente mejor valor, también asigna a la mayoría de nodos la misma comunidad. Con Label Propagation y Radicchi Strong pasa parecido, por lo que nos quedamos con CoDA, con una separabilidad de 0,3963 y una varianza bastante pequeña.

Por tanto, teniendo en cuenta los resultados de 5.10 y viendo las divisiones que han hecho estos algoritmos en la tabla B.7, se pueden tomar CONGA como mejor algoritmo en conductancia, Multilevel en cohesividad y CoDA en separabilidad.

5.3.2. Respuestas

En la tabla 5.11, se ve que Bigclam y Radicchi Strong obtienen valores perfectos y los siguientes mejores resultados son de Clauset Newman Moore, CONGA y CONGO. Cabe destacar los buenos resultados en cohesividad de Fastgreedy, Multilevel y Walktrap, que quedan cerca de Clauset Newman Moore.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Bigclam	*0.0000*	0.0000	*1.0000*	0.0000	*0.0000*	0.0000
Clauset Newman Moore	0.1914	0.0316	**0.9985**	0.0001	2.1656	12.8855
Coda	0.8967	0.0365	0.2432	0.0375	0.0448	0.0043
Conga	0.0002	0.0000	0.9144	0.0043	**0.0078**	0.0014
Congo	**0.0001**	0.0000	0.9117	0.0045	0.0089	0.0017
Fastgreedy	0.1824	0.0311	0.9976	0.0002	2.3390	14.8263
Infomap	0.0579	0.0197	0.7906	0.0658	0.7283	5.4660
Label Propagation	0.0455	0.0182	0.7537	0.0687	2.2694	136.5653
Leading Eigenvector	0.1155	0.0201	0.9012	0.0353	3.6110	55.1834
Multilevel	0.1824	0.0311	0.9976	0.0002	2.3390	14.8263
Radicchi Strong	*0.0000*	0.0000	0.7121	0.0735	*0.0000*	0.0000
Radicchi Weak	0.1493	0.0863	0.8395	0.0515	0.4981	1.6850
Spinglass	0.4665	0.0437	0.9846	0.0043	2.9522	4.1055
Walktrap	0.1824	0.0311	0.9971	0.0002	2.3386	14.8273

Cuadro 5.11: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Respuestas. Mejores resultados en negrita

Observando la tabla B.8 se puede ver que Bigclam, y Radicchi Strong dividen la mayoría de redes ego en una o ninguna comunidades, por lo que no las tenemos en cuenta.

Se tiene, entonces, que CONGO obtiene los mejores resultados en conductancia, Clauset Newman Moore en cohesividad y CONGA en separabilidad, ya que estos algoritmos sí dividen un número considerable de redes en comunidades.

5.3.3. Retweets

En la tabla 5.12, se puede ver que se obtienen valores perfectos en Bigclam y Radicchi Strong y los siguientes mejores resultados son de CONGO y Multilevel.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Bigclam	*0.0000*	0.0000	*1.0000*	0.0000	*0.0000*	0.0000
Clauset Newman Moore	0.1380	0.0276	0.9582	0.0158	5.9639	382.5303
Coda	0.8526	0.0456	0.2998	0.0524	4.6448	935.3134
Conga	0.0022	0.0000	0.9409	0.0069	0.3193	1.0861
Congo	**0.0012**	0.0000	0.9543	0.0057	**0.1580**	0.4653
Fastgreedy	0.1195	0.0231	0.9690	0.0118	4.9368	316.6836
Infomap	0.0391	0.0103	0.8456	0.0616	3.8070	316.8927
Label Propagation	0.0162	0.0050	0.7912	0.0739	0.6572	10.9916
Leading Eigenvector	0.1058	0.0207	0.9561	0.0148	4.7125	314.6055
Multilevel	0.1196	0.0232	**0.9700**	0.0112	4.9362	316.6884
Radicchi Strong	*0.0000*	0.0000	0.7670	0.0803	*0.0000*	0.0000
Radicchi Weak	0.0823	0.0248	0.9005	0.0346	1.4719	17.7656
Spinglass	0.4790	0.0908	0.9461	0.0185	2.6089	6.7053
Walktrap	0.1080	0.0236	0.9538	0.0181	4.9088	316.9378

Cuadro 5.12: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de *Retweets*. Mejores resultados en negrita

De nuevo, mirando la B.9 se observa que Bigclam y Radicchi Strong apenas hacen divisiones en comunidades, por lo que se han descartado.

Los mejores algoritmos que sí han hecho división de las redes en comunidades son CONGO, con mejor resultado en conductancia y separabilidad, y Multilevel en cohesividad.

5.3.4. Menciones, respuestas y retweets

En la tabla 5.13, se puede ver que Clique Percolation obtiene unos resultados óptimos en las métricas y los siguientes mejores resultados los obtienen CONGO y Multilevel.

Algorithm	**Media conduc.**	**Varianza conduc.**	**Media cohesiv.**	**Varianza cohesiv.**	**Media separa.**	**Varianza separa.**
Clauset Newman Moore	0.2623	0.0336	0.9276	0.0181	2.4140	26.3821
Clique Percolation	*0.0000*	0.0000	0.8889	0.0247	*0.0000*	0.0000
Coda	0.7355	0.0779	0.3937	0.0498	0.5319	5.5606
Conga	0.0125	0.0056	0.8868	0.0129	0.3997	1.4720
Congo	**0.0098**	0.0053	0.8862	0.0129	**0.3606**	1.5737
Fastgreedy	0.2677	0.0339	0.9390	0.0153	2.1298	21.2711
Infomap	0.0924	0.0290	0.7130	0.0697	1.2567	24.6315
Label Propagation	0.0433	0.0180	0.6269	0.0695	4.3785	423.2602
Leading Eigenvector	0.2081	0.0337	0.8525	0.0489	3.4469	91.2723
Multilevel	0.2707	0.0349	**0.9499**	0.0123	2.1249	21.2912
Radicchi Strong	0.0014	0.0004	0.5924	0.0681	0.0064	0.0083
Radicchi Weak	0.2648	0.1164	0.8291	0.0479	1.0090	14.2768
Spinglass	0.4241	0.0402	0.9258	0.0218	2.0545	4.1854
Walktrap	0.2701	0.0463	0.9408	0.0197	2.0831	21.4265

Cuadro 5.13: Valores medios y de varianza de la ejecución de cada uno de los algoritmos con el conjunto de datos de Menciones, Respuestas y *Retweets*. Mejores resultados en negrita

En la tabla B.11 se observa que Clique Percolation no realiza muchas divisiones en comunidades, que es lo que hace que se tengan esos valores óptimos.

Por tanto, se tiene que CONGO es el mejor en conductancia y separabilidad y Multilevel en cohesividad.

5.3.5. Conclusiones

Como se ha visto, los algoritmos que menos divisiones han hecho han sido Bigclam, Clique Percolation y Radicchi Strong, como ya ocurrió en la otra ejecución. Las razones a las que puede deberse se describieron en el apartado 5.2.6.

A continuación, en la figura 5.4 se muestran de nuevo tres gráficas en las que se va a poder comparar el resultado de las métricas en cada uno de los conjuntos de datos: Menciones (Men), Respuestas (Res), Retweets (Ret), Menciones, Respuestas y Retweets (MRR) Follows (Fol).

Observando las gráficas de 5.4, se ve que el conjunto de datos que obtiene mejores resultados en las tres métricas es el formado por Respuestas, seguido del de *Retweets*. Los algoritmos con los que se han conseguido estos resultados son: CONGO en conductividad, Clauset Newman Moore en cohesividad y CONGA en separabilidad. Pero, hay que destacar que Multilevel ha obtenido el mejor resultado en los otros conjuntos de datos, quedando muy cerca del Clauset Newman Moore en Respuestas.

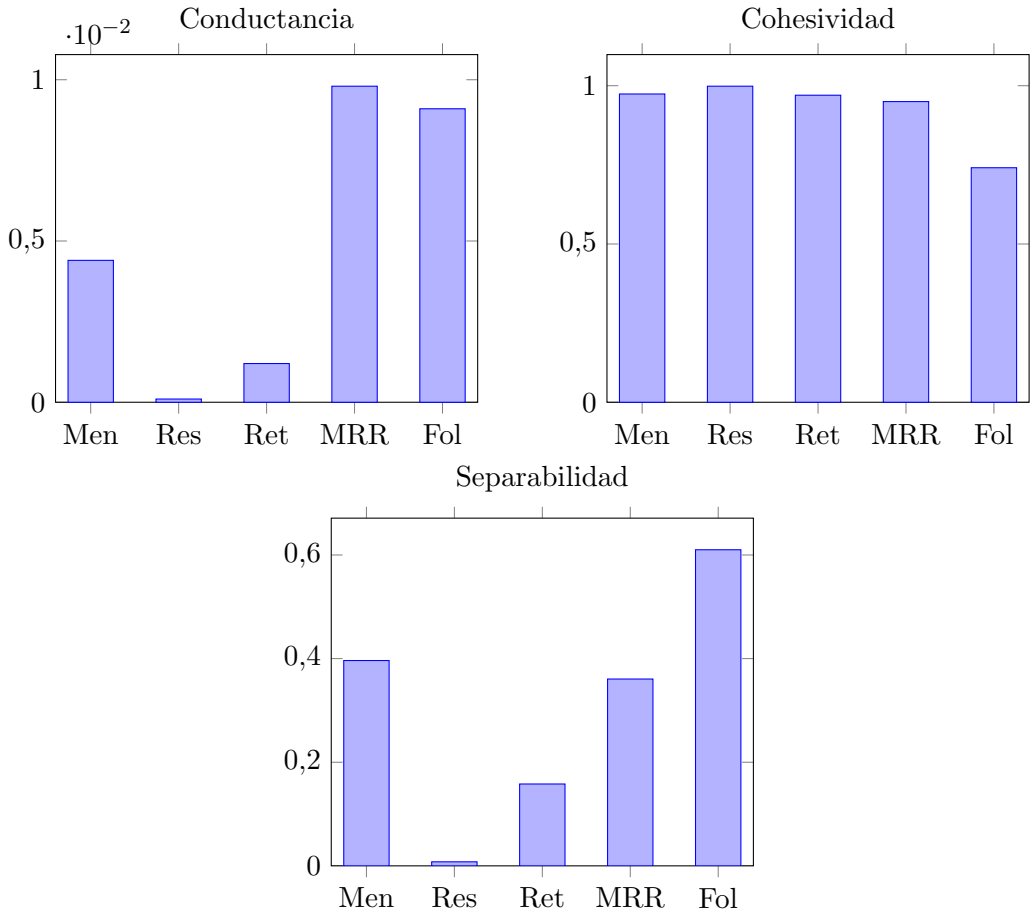

Figura 5.4: Gráficas que muestran los mejores resultados que han obtenido los cinco conjuntos de datos en cada una de las métricas haciendo un filtro de comunidades de más de 8 nodos. Arriba izquierda: conductancia, arriba derecha: cohesividad y abajo, separabilidad. Conjuntos de datos: Menciones (Men), Respuestas (Res), Retweets (Ret), Menciones, Respuestas y Retweets (MRR) Follows (Fol)

Cabe destacar que la tabla de resultados de conductancia muestra sólo valores de 0 a 0,01. Esta es la causa de que el resultado del conjunto formado por Menciones, Respuestas y *Retweets* se aleje "tanto" de los resultados de cada interacción por separado. En realidad todos los modelos están obteniendo una buena conductancia baja.

5.4. Representación gráfica

Los algoritmos de detección de comunidades se encargan de asignar a cada nodo una comunidad, y esto puede representarse gráficamente como se va a ver a continuación. Esta representación de grafos se ha realizado mediante Gephi, un *software* de análisis de redes y visualización.

En la figura 5.5 se puede ver la red formada por las 7 redes ego que se han estudiado en el conjunto de datos de *Follows*. Los nodos coloreados son los *egos*.

De esa red se ha seleccionado la red ego del nodo coloreado de abajo a la derecha, el usuario *Al_harbi_khaled* y, a continuación, se van a mostrar diferentes soluciones de los algoritmos de detección de comunidades. En las figuras 5.6 y 5.7 muetran las soluciones de los algoritmos CONGO y Spinglass, que fueron los que mejores resultados en métricas dieron. Después se muestra otra solución de Label Propagation en la figura 5.8.

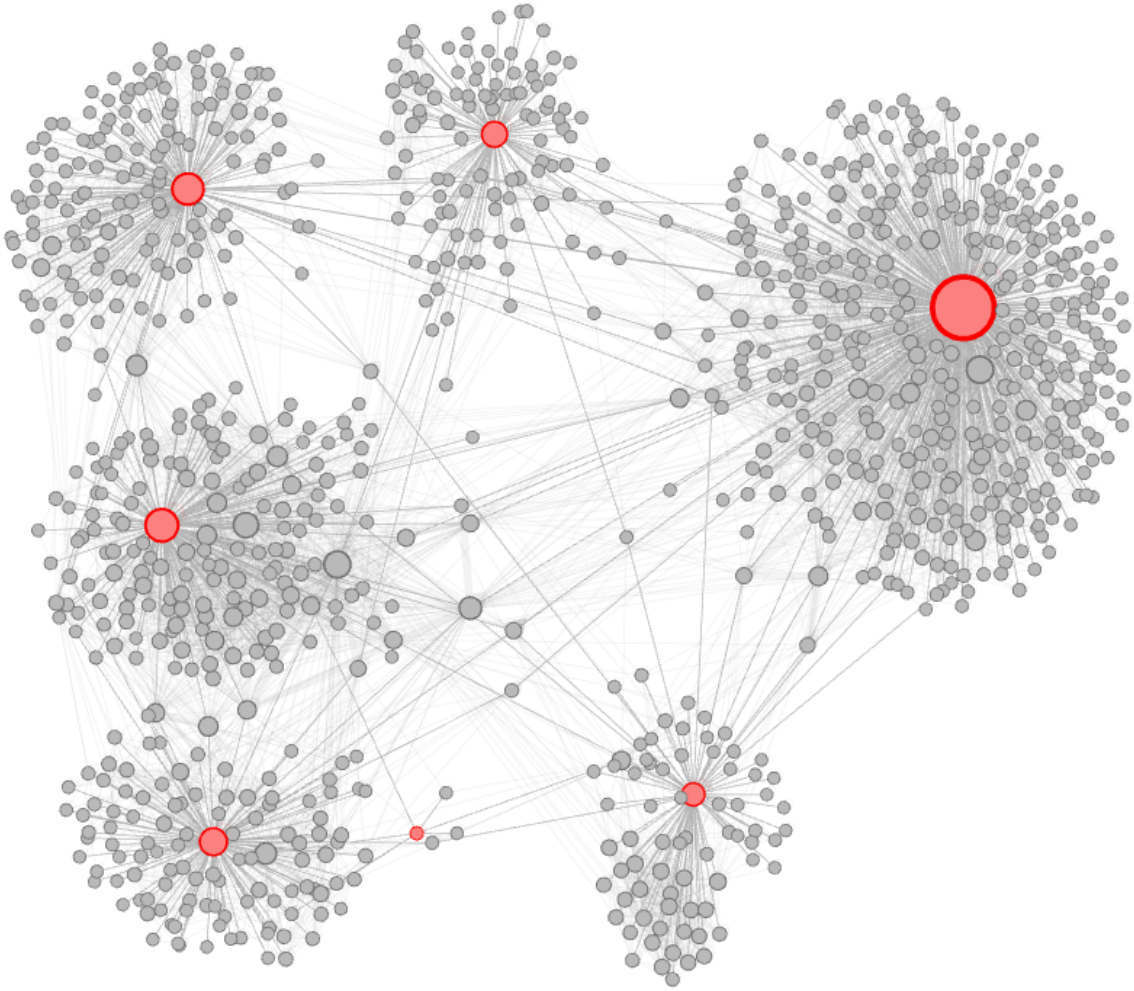

Figura 5.5: Red que foman las redes ego del conjunto de datos de *Follows*. Los nodos coloreados son los *ego*

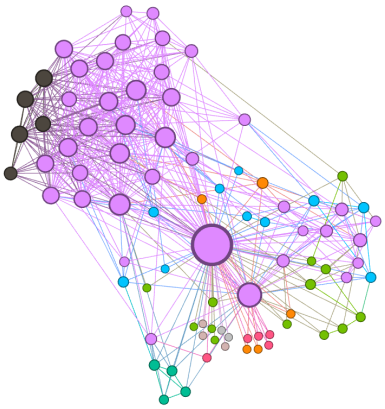

Figura 5.6: Red ego de *Al_harbi_khaled* dividida en comunidades por CONGO

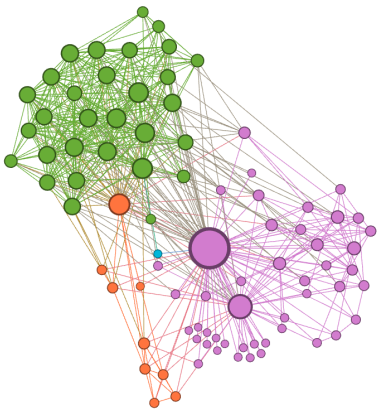

Figura 5.7: Red ego de *Al_harbi_khaled* dividida en comunidades por Spinglass

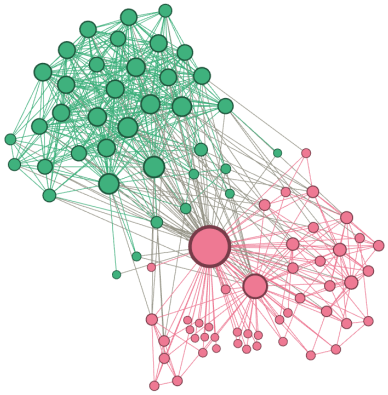

Figura 5.8: Red ego de *Al_harbi_khaled* dividida en comunidades por el algoritmo Label Propagation

Como se comentó con anterioridad, las redes ego de Menciones, Respuestas, *Retweets* tienen, en general, una estructura estrellada. A continuación se van a mostrar algunos ejemplos de este tipo comunidades y su división en comunidades por los algoritmos que dieron mejores resultados con el conjunto de datos de Respuestas: CONGO, Clauset Newman Moore y CONGA.

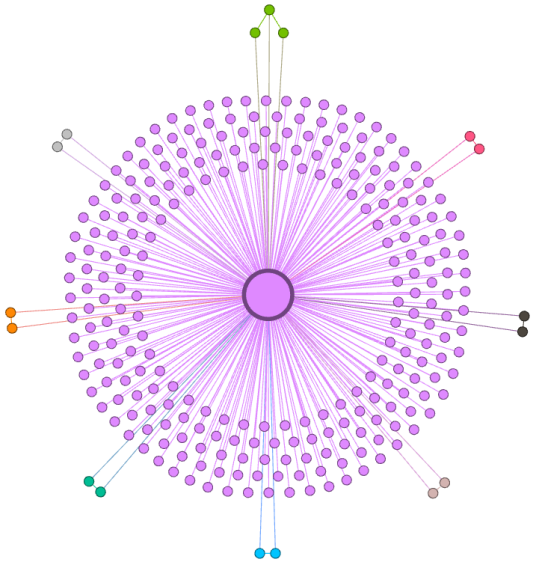

Figura 5.9: Red ego de Respuestas dividida en comunidades por Clauset Newman Moore

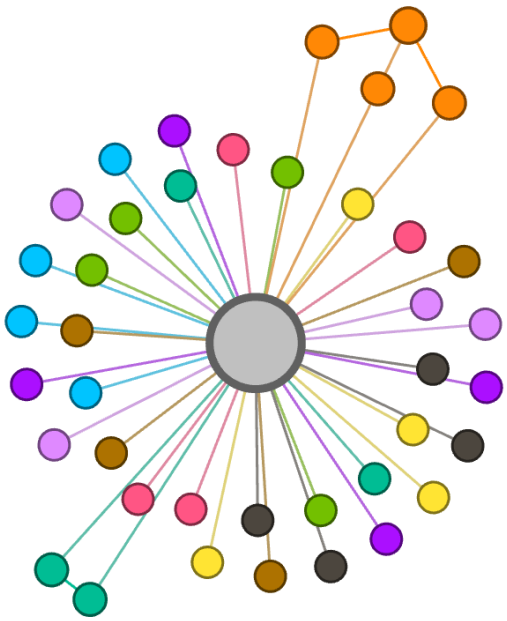

Figura 5.10: Red ego de Respuestas dividida en comunidades por CONGO

CONGA y CONGO dan resultados muy parecidos, por lo que se ha omitido su comparación. Las figuras 5.11 y 5.12 muestran las distintas divisiones en comunidades que le dan a una misma comunidad.

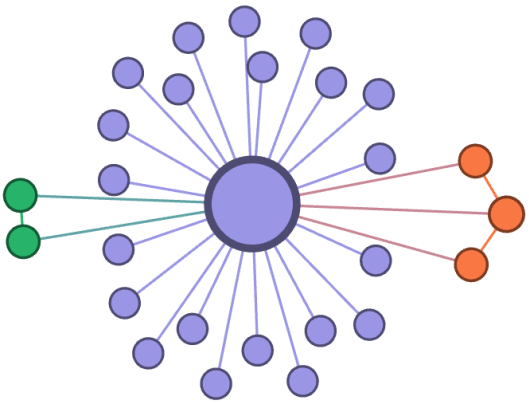

Figura 5.11: Red ego de Respuestas dividida en comunidades por Clauset Newman Moore

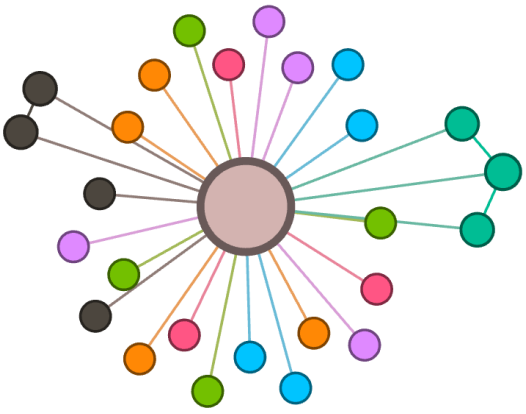

Figura 5.12: Red ego dividida en comunidades por CONGA

Como se puede apreciar, CONGA y CONGO separan en muchas comunidades los nodos sueltos que solo están unidos al *ego*, mientras que Clauset Newman Moore incluye a todos estos en una misma comunidad. Los tres algoritmos suelen separar los grupos enlazados entre sí a parte.

6

Conclusiones y trabajo futuro

Tras haber realizado este trabajo, podemos sacar varias conclusiones que se van a detallar a continuación. A demás, han surgido nuevas incógnitas que se describen en la sección de Trabajo Futuro.

6.1. Conclusiones

La detección de comunidades es un problema que se puede aplicar en muchos ámbitos y no solo para el análisis de redes sociales que es para lo que se ha utilizado en este trabajo. Pero, cabe destacar su particular importancia en el análisis de SNs ya que en ellas todo el mundo comparte información personal que incluye gustos, fotos, opiniones, círculo de amigos... Y todo esto puede ser analizado para extraer conclusiones.

Este trabajo se ha centrado en el análisis de los algoritmos que se utilizan para la detección de comunidades y el planteamiento de modelos de datos alternativos al estándar *following-follower*. Estos modelos se basan en incluir Menciones, Respuestas y *Retweets* a la hora de representar la red con grafos. Cabe destacar, en particular, los buenos resultados de los conjuntos de datos compuestos por Respuestas y *Retweets*.

Analizando los datos, se ha visto que las redes ego que se han analizado tienen una estructura estrellada, que puede resultar más complicado de dividir en comunidades. A pesar de esto, los modelos alternativos han dado mejores resultados en cuanto a la evaluación de la conductancia, cohesividad y separabilidad de las comunidades que extraían. Por otro lado, los algoritmos que mejores resultados han obtenido son CONGO y CONGA. Y los que peores han resultado han sido los que se basaban en cliques o conjuntos de nodos. Esto puede deberse a que, al tratarse de redes pequeñas o con forma estrellada, no aparecían suficientes elementos de estos tipos como para dividir la red en comunidades.

Por último, es importante resaltar la forma estrellada en las redes obtenidas mediante las Menciones, Respuestas y *Retweets*. Esta estructura concuerda con la idea que se planteó al principio de la investigación: existe una persona, el *ego*, cuya intención es la de reclutar gente, por eso interactúa con otro usuarios, aunque estos no suelen interactuar entre sí. Esta falta de interacción entre los *alters* puede deberse a una medida de discreción o, simplemente, a que no se conozcen entre sí.

6.2. Trabajo futuro

Finalmente, quedan varias cuestiones que podrían ampliarse en un trabajo futuro:

- Realizar el estudio llevado a cabo en este trabajo con otro conjunto de datos más contrastado para probar más los modelos alternativos que se han planteado y ver si son realmente mejor en otros ámbitos. Puesto que el objetivo sería probar los modelos formados por interacciones del tipo respuestas o *retweets*, estos conjuntos de datos tendrían que obtenerse de una red social como Twitter, que es la que se ha utilizado aquí.

  Existen muchos problemas que pueden modelarse como grafos y, puesto que la detección de comunidades es un problema de *clustering* en grafos, podría aplicarse en ámbitos como por ejemplo: detección de comunidades antivacunas, *marketing*, sistemas de recomendación (agrupando comunidades de consumidores) o política.

- Probar la ejecución con otros algoritmos para mejorar el estado del arte y ver si se pueden mejorar los resultados obtenidos con los aquí presentados. Estos algoritmos podrían estar basados en computación evolutiva, en MOGA o ser algoritmos de enjambre para la detección de comunidades (ACO, PSO, ABC, ...).

Glosario de acrónimos

- **ABC**: Artificial Bee Colony
- **ACO**: Ant Colony Optimization
- **API**: Application Programming Interface
- **BIGCLAM**: Cluster Affilitation Model for Big Networks
- **CoDA**: Communities through Directed Affiliations
- **CDP**: Community Detection Problem
- **CONGA**: Cluster-Overlapping Newman Girvan Algorithm
- **CONGO**: CONGA Optimized
- **EBC**: Edge Betweenness Centrality
- **ISIS**: Islamic State in Iraq and Syria
- **JSON**: JavaScript Object Notation
- **MOGA**: Multi-Objective Genetic Algorithm
- **PSO**: Particle Swarm Optimization
- **SQL**: Structured Query Language
- **SN**: Social Network
- **SNA**: Social Network Analysis
- **SNAP**: Stanford Network Analysis Project
- **SP**: Shortest Paths

Bibliografía

[1] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pp. 587–596, 2013.

[2] R. Zafarani, M. A. Abbasi, and H. Liu, *Social Media Mining: An Introduction*. Cambridge University Press, 2014.

[3] J. Yang, J. McAuley, and J. Leskovec, "Detecting cohesive and 2-mode communities indirected and undirected networks," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, (New York, NY, USA), pp. 323–332, Association for Computing Machinery (ACM), 2014.

[4] S. Gregory, "An algorithm to find overlapping community structure in networks," in *Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, September 17-21, 2007, Proceedings*, pp. 91–102, 2007.

[5] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[6] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, p. 036106, Sep 2007.

[7] G. B. Orgaz and D. Camacho, "Evolutionary clustering algorithm for community detection using graph-based information," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*, pp. 930–937, 2014.

[8] R. Lara-Cabrera, C. Cotta, and A. Fernández-Leiva, "An analysis of the structure and evolution of the scientific collaboration network of computer intelligence in games," *Physica A: Statistical Mechanics and its Applications*, vol. 395, pp. 523–536, 2014.

[9] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 97–107, Jan. 2014.

[10] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks.," in *Neural Information Processing Systems* (P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 548–556, 2012.

[11] A. González-Pardo, J. J. Jung, and D. Camacho, "Aco-based clustering for ego network analysis," *Future Generation Computer Systems*, vol. 66, pp. 160–170, 2017.

[12] J. Yang, J. J. McAuley, and J. Leskovec, "Community detection in networks with node attributes," in *ICDM*, pp. 1151–1156, IEEE Computer Society, 2013.

[13] "Neo4j," *https://neo4j.com/*.

[14] P. Mazzuca and Y. T, "Circulo," *http://www.lab41.org/circulo-a-community-detection-evaluation-framework/*, 2015.

[15] J. A. McHugh, *Algorithmic Graph Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990.

[16] R. Kannan, S. Vempala, and A. Vetta, "On clusterings: Good, bad and spectral," *Journal of the Association for Computing Machinery (ACM)*, vol. 51, no. 3, pp. 497–515, 2004.

[17] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[18] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75 – 174, 2010.

[19] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, pp. 8577–8582, June 2006.

[20] A. Clauset, M. E. J. Newman, , and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, pp. 1–6, 2004.

[21] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, vol. E 69, no. 026113, 2004.

[22] M. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E*, vol. 69, September 2003.

[23] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, pp. 814–818, June 2005.

[24] S. Gregory, "A fast algorithm to find overlapping communities in networks," in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, pp. 408–423, 2008.

[25] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, no. 3, p. 036104, 2006.

[26] Z. Yang, R. Algesheimer, and C. J. Tessone, "A comparative analysis of community detection algorithms on artificial networks," *Scientific Reports*, vol. abs/1608.00763, 2016.

[27] H. Djidjev, "A scalable multilevel algorithm for graph clustering and community structure detection," in *WAW*, vol. 4936 of *Lecture Notes in Computer Science*, pp. 117–128, Springer, 2006.

[28] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. 9, p. 2658, 2004.

[29] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, June 1998.

[30] J. Reichardt and S. Bornholdt, "Statistical mechanics of community detection," *Physical Review E*, vol. 74, p. 016110, 2006.

[31] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006.

[32] "Twitter developer," *https://developer.twitter.com/en/docs*.

[33] "Tweepy. librería de python," *http://www.tweepy.org/*.

[34] "Twython. librería de python," *https://twython.readthedocs.io/en/latest/*.

[35] J. Bonneau, J. Anderson, R. J. Anderson, and F. Stajano, "Eight friends are enough: social graph approximation via public listings," in *SNS*, pp. 13–18, Association for Computing Machinery (ACM), 2009.

[36] G. B. Orgaz, J. C. Hernandez-Castro, and D. Camacho, "Detecting discussion communities on vaccination in twitter," *Future Generation Computer Systems*, vol. 66, pp. 125–136, 2017.

[37] "Cypher," *https://neo4j.com/developer/cypher-query-language/*.

[38] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.

[39] "Snap(stanford network analysis project)," *http://snap.stanford.edu/*.

[40] "Py2neo," *http://py2neo.org/v3/*.

A

Respuesta de Twitter

A continuación se muestra un ejemplo de respuesta JSON que devuelve la API de Twitter tras una consulta.


```
{
  "statuses": [
    {
      "coordinates": null,
      "favorited": false,
      "truncated": false,
      "created_at": "Mon Sep 24 03:35:21 +0000 2012",
      "id_str": "250075927172759552",
      "entities": {
        "urls": [

        ],
        "hashtags": [
          {
            "text": "freebandnames",
            "indices": [
              20,
              34
            ]
          }
        ],
        "user_mentions": [

        ]
      },
      "in_reply_to_user_id_str": null,
      "contributors": null,
      "text": "Aggressive Ponytail #freebandnames",
      "metadata": {
        "iso_language_code": "en",
```

```
    "result_type": "recent"
  },
  "retweet_count": 0,
  "in_reply_to_status_id_str": null,
  "id": 250075927172759552,
  "geo": null,
  "retweeted": false,
  "in_reply_to_user_id": null,
  "place": null,
  "user": {
    "profile_sidebar_fill_color": "DDEEF6",
    "profile_sidebar_border_color": "C0DEED",
    "profile_background_tile": false,
    "name": "Sean Cummings",
    "profile_image_url": "http://a0.twimg.com/profile_images/2359746665/1v6z
    "created_at": "Mon Apr 26 06:01:55 +0000 2010",
    "location": "LA, CA",
    "follow_request_sent": null,
    "profile_link_color": "0084B4",
    "is_translator": false,
    "id_str": "137238150",
    "entities": {
      "url": {
        "urls": [
          {
            "expanded_url": null,
            "url": "",
            "indices": [
              0,
              0
            ]
          }
        ]
      },
      "description": {
        "urls": [

        ]
      }
    },
    "default_profile": true,
    "contributors_enabled": false,
    "favourites_count": 0,
    "url": null,
    "profile_image_url_https": "https://si0.twimg.com/profile_images/2359746
    "utc_offset": -28800,
    "id": 137238150,
    "profile_use_background_image": true,
    "listed_count": 2,
    "profile_text_color": "333333",
    "lang": "en",
```

```
        "followers_count": 70,
        "protected": false,
        "notifications": null,
        "profile_background_image_url_https": "https://si0.twimg.com/images/them
        "profile_background_color": "C0DEED",
        "verified": false,
        "geo_enabled": true,
        "time_zone": "Pacific Time (US & Canada)",
        "description": "Born 330 Live 310",
        "default_profile_image": false,
        "profile_background_image_url": "http://a0.twimg.com/images/themes/theme
        "statuses_count": 579,
        "friends_count": 110,
        "following": null,
        "show_all_inline_media": false,
        "screen_name": "sean_cummings"
      },
      "in_reply_to_screen_name": null,
      "source": "Twitter for Mac",
      "in_reply_to_status_id": null
    }
    ],
  "search_metadata": {
    "max_id": 250126199840518145,
    "since_id": 24012619984051000,
    "refresh_url": "?since_id=250126199840518145&q=%23freebandnames&result_type=
    "next_results": "?max_id=249279667666817023&q=%23freebandnames&count=4&inclu
    "count": 4,
    "completed_in": 0.035,
    "since_id_str": "24012619984051000",
    "query": "%23freebandnames",
    "max_id_str": "250126199840518145"
  }
}
```

B

Tablas

B.1. Primera ejecución

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 233, 1: 117, 2: 1, 5: 1}
Clauset Newman Moore	{0: 19, 1: 271, 2: 27, 3: 1, 5: 1, 7: 1, 14: 1}
Clique Percolation	{0: 121, 1: 89, 2: 2, 13: 1}
Coda	{0: 137, 1: 77, 2: 5, 3: 3, 4: 95, 5: 4, 6: 4, 7: 9, 8: 5, 9: 4, 10: 1, 11: 3, 12: 4, 14: 3, 15: 1, 18: 2, 19: 2, 22: 1, 24: 2, 25: 2, 27: 1, 35: 1, 41: 1}
Conga	{32: 1, 1: 122, 2: 8, 3: 4, 4: 3, 8: 2, 10: 1, 34: 1}
Congo	{1: 122, 2: 8, 3: 4, 4: 3, 8: 2, 10: 1, 34: 1, 30: 1}
Fastgreedy	{1: 345, 2: 17, 3: 2, 5: 1, 13: 1, 14: 1}
Infomap	{1: 365, 14: 1, 15: 1}
Label Propagation	{1: 358, 2: 9}
Leading Eigenvector	{8: 1, 1: 346, 2: 17, 3: 2}
Multilevel	{1: 329, 2: 31, 3: 4, 5: 1, 13: 1, 14: 1}
Radicchi Strong	{1: 367}
Radicchi Weak	{1: 361, 2: 3, 4: 1, 21: 1, 14: 1}
Spinglass	{1: 310, 2: 47, 3: 5, 4: 2, 6: 1, 12: 1, 13: 1}
Walktrap	{1: 348, 2: 14, 3: 2, 5: 1, 12: 1, 14: 1}

Cuadro B.1: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 510, 1: 275, 2: 2, 35: 1, 5: 1, 9: 1, 12: 1, 34: 1, 3: 2, 21: 1, 29: 1}
Clauset Newman Moore	{0: 14, 1: 648, 2: 57, 3: 5, 4: 9, 5: 1, 9: 1, 16: 1, 25: 1}
Clique Percolation	{0: 323, 1: 209, 2: 4, 3: 3, 4: 1, 8: 1, 24: 1}
Coda	{0: 284, 1: 162, 2: 8, 3: 9, 4: 216, 5: 10, 6: 4, 7: 21, 8: 7, 9: 13, 10: 6, 11: 17, 12: 9, 13: 6, 14: 2, 15: 2, 16: 3, 17: 1, 18: 2, 19: 2, 20: 3, 22: 3, 23: 1, 25: 3, 27: 1, 28: 1, 31: 1, 35: 1, 46: 1, 56: 1, 63: 1}
Conga	{1: 265, 2: 15, 3: 3, 4: 2, 5: 3, 6: 2, 7: 3, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 2, 14: 3, 15: 2, 20: 1, 21: 1, 29: 1, 31: 2, 34: 1, 37: 2, 41: 1, 69: 1, 91: 1, 121: 1}
Congo	{1: 265, 2: 15, 3: 3, 4: 2, 5: 3, 6: 2, 7: 3, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 2, 14: 3, 15: 2, 20: 1, 21: 1, 29: 1, 31: 2, 34: 1, 37: 2, 41: 1, 69: 1, 89: 1, 121: 1}
Fastgreedy	{1: 736, 2: 42, 3: 8, 4: 9, 5: 1, 7: 1, 9: 2, 16: 1, 25: 1}
Infomap	{1: 790, 2: 1, 3: 2, 4: 2, 5: 1, 7: 1, 9: 2, 19: 1, 25: 1}
Label Propagation	{1: 778, 2: 23}
Leading Eigenvector	{1: 745, 2: 46, 3: 6, 4: 2}
Multilevel	{1: 712, 2: 66, 3: 8, 4: 9, 5: 1, 7: 1, 9: 2, 16: 1, 25: 1}
Radicchi Strong	{1: 801}
Radicchi Weak	{1: 781, 2: 7, 4: 5, 6: 3, 8: 1, 12: 1, 46: 1, 16: 1, 22: 1}
Spinglass	{1: 667, 2: 94, 3: 11, 4: 10, 5: 8, 7: 2, 8: 3, 9: 2, 10: 1, 11: 1, 15: 1, 17: 1}
Walktrap	{1: 738, 2: 39, 3: 9, 4: 9, 5: 1, 7: 1, 9: 2, 17: 1, 25: 1}

Cuadro B.2: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Respuestas

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 768, 1: 74, 2: 8, 3: 1, 39: 2, 72: 1, 9: 1, 12: 1, 13: 1, 45: 1, 119: 1, 29: 1}
Clauset Newman Moore	{0: 27, 1: 646, 2: 60, 3: 7, 4: 1, 5: 3}
Clique Percolation	{0: 532, 1: 69, 2: 6, 3: 1, 4: 1}
Coda	{0: 402, 1: 256, 2: 14, 3: 2, 4: 39, 5: 19, 6: 16, 7: 24, 8: 7, 9: 4, 10: 6, 11: 9, 12: 8, 13: 2, 14: 4, 15: 3, 16: 4, 17: 2, 18: 1, 19: 1, 20: 4, 21: 1, 22: 1, 23: 3, 24: 1, 25: 1, 26: 3, 27: 2, 28: 1, 29: 2, 30: 1, 32: 1, 33: 1, 37: 1, 38: 1, 40: 1, 41: 4, 42: 1, 43: 1, 45: 1, 48: 1, 49: 1, 57: 1}
Conga	{1: 33, 2: 17, 3: 8, 4: 3, 5: 4, 6: 2, 7: 3, 8: 3, 10: 2, 11: 1, 12: 2, 13: 2, 14: 2, 15: 1, 19: 2, 20: 1, 21: 2, 22: 2, 27: 1, 31: 2, 32: 2, 162: 1, 44: 1, 63: 1, 67: 1, 72: 1, 87: 1, 102: 2, 114: 1}
Congo	{1: 33, 2: 17, 3: 8, 4: 3, 5: 4, 6: 1, 7: 5, 8: 3, 10: 2, 11: 1, 12: 2, 13: 1, 14: 2, 15: 1, 19: 2, 20: 1, 21: 2, 22: 2, 27: 2, 30: 1, 31: 1, 32: 1, 162: 1, 44: 1, 63: 1, 68: 1, 72: 1, 87: 1, 102: 2, 114: 1}
Fastgreedy	{1: 790, 2: 62, 3: 6, 4: 1, 5: 3}
Infomap	{1: 842, 2: 10, 3: 8, 5: 1, 7: 1}
Label Propagation	{1: 846, 2: 14, 3: 2}
Leading Eigenvector	{1: 793, 2: 61, 3: 2, 4: 1, 5: 3}
Multilevel	{1: 781, 2: 71, 3: 6, 4: 1, 5: 2, 6: 1}
Radicchi Strong	{1: 862}
Radicchi Weak	{1: 819, 2: 37, 4: 4, 5: 1, 6: 1}
Spinglass	{1: 709, 2: 96, 3: 12, 4: 8, 5: 6, 6: 2, 7: 3, 8: 6, 10: 1, 11: 1, 13: 1, 14: 2, 16: 2, 17: 1, 19: 2, 21: 4, 22: 2, 23: 1, 24: 1, 25: 2}
Walktrap	{1: 799, 2: 53, 3: 5, 4: 2, 5: 2, 12: 1}

Cuadro B.3: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Retweets

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 1194, 1: 1097, 2: 14, 3: 3, 4: 3, 5: 6, 6: 2, 7: 1, 8: 1, 9: 1, 10: 4, 11: 1, 12: 1, 13: 1, 19: 1, 30: 1}
Clauset Newman Moore	{0: 38, 1: 1833, 2: 218, 3: 19, 68: 1, 5: 5, 6: 3, 7: 2, 8: 5, 9: 1, 10: 2, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 2, 55: 1, 4: 14, 26: 1, 27: 1}
Clique Percolation	{0: 736, 1: 960, 2: 30, 3: 9, 32: 1, 5: 2, 6: 2, 7: 3, 8: 1, 9: 1, 10: 1, 11: 1, 14: 1, 16: 1, 67: 1, 25: 1, 4: 1, 19: 2}
Coda	{0: 575, 1: 368, 2: 18, 3: 25, 4: 882, 5: 42, 6: 26, 7: 59, 8: 22, 9: 29, 10: 17, 11: 73, 12: 29, 13: 5, 14: 9, 15: 24, 16: 6, 17: 5, 18: 9, 19: 4, 20: 9, 21: 6, 22: 25, 23: 5, 24: 3, 25: 24, 26: 5, 27: 1, 28: 1, 30: 1, 33: 1, 35: 1, 36: 1, 37: 3, 39: 1, 40: 1, 42: 2, 43: 5, 45: 1, 47: 1, 48: 2, 50: 2, 52: 1, 54: 1, 57: 1, 61: 1, 63: 1, 64: 1, 87: 1}
Conga	{1: 1145, 2: 69, 3: 28, 4: 12, 5: 7, 6: 4, 7: 4, 8: 4, 9: 2, 10: 2, 139: 1, 12: 2, 13: 2, 14: 2, 15: 3, 16: 1, 17: 2, 19: 2, 20: 3, 149: 1, 22: 1, 23: 1, 24: 1, 25: 3, 26: 1, 32: 2, 33: 2, 34: 1, 35: 2, 165: 1, 169: 1, 42: 1, 45: 1, 46: 1, 47: 1, 151: 1, 66: 1, 11: 1, 69: 1, 71: 1, 73: 1, 141: 1, 85: 1, 143: 1, 92: 1, 96: 2, 144: 1, 231: 1, 116: 1, 121: 1, 21: 3}
Congo	{128: 1, 1: 1145, 2: 69, 3: 28, 4: 13, 5: 7, 6: 5, 7: 4, 8: 3, 9: 2, 10: 3, 151: 1, 12: 3, 141: 1, 14: 1, 15: 3, 144: 1, 17: 2, 19: 2, 20: 3, 21: 2, 22: 1, 23: 1, 24: 2, 25: 2, 26: 2, 30: 1, 32: 1, 33: 2, 34: 1, 35: 1, 165: 1, 169: 1, 42: 1, 45: 1, 46: 1, 47: 1, 138: 1, 66: 1, 68: 1, 72: 1, 73: 1, 13: 2, 85: 1, 92: 1, 96: 2, 16: 1, 230: 1, 116: 1, 121: 1, 149: 1}
Fastgreedy	{1: 2109, 2: 155, 3: 29, 4: 15, 5: 6, 6: 3, 7: 3, 8: 5, 9: 2, 10: 1, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 2, 55: 1, 68: 1, 33: 1, 28: 1, 26: 1}

Cuadro B.4: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones, Respuestas y Retweets. Primera mitad

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Infomap	{1: 2293, 2: 5, 3: 7, 4: 6, 5: 2, 6: 2, 7: 3, 8: 5, 9: 1, 10: 1, 11: 3, 12: 2, 15: 1, 16: 1, 17: 1, 20: 1, 22: 1, 30: 1, 33: 1, 36: 1, 55: 1, 78: 1}
Label Propagation	{1: 2275, 2: 63, 3: 1, 4: 1}
Leading Eigenvector	{1: 2125, 2: 166, 3: 28, 4: 15, 5: 1}
Multilevel	{1: 2002, 2: 259, 3: 28, 4: 18, 5: 7, 6: 3, 7: 2, 8: 5, 9: 3, 10: 1, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 2, 33: 1, 55: 1, 68: 1, 26: 1, 28: 1}
Radicchi Strong	{1: 2340}
Radicchi Weak	{1: 2241, 2: 45, 4: 17, 6: 9, 7: 1, 8: 5, 10: 3, 12: 2, 14: 3, 16: 2, 18: 1, 19: 1, 21: 1, 23: 1, 28: 1, 30: 1, 32: 1, 38: 1, 50: 1, 57: 1, 89: 1, 92: 1}
Spinglass	{1: 1914, 2: 316, 3: 38, 4: 24, 5: 9, 6: 10, 7: 3, 8: 4, 9: 1, 10: 2, 11: 2, 13: 2, 16: 1, 17: 1, 18: 1, 19: 3, 20: 2, 21: 3, 22: 1, 24: 3}
Walktrap	{1: 2124, 2: 131, 3: 29, 4: 17, 5: 7, 6: 5, 7: 4, 8: 6, 9: 2, 10: 3, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 1, 21: 1, 27: 1, 30: 1, 33: 1, 55: 1, 74: 1}

Cuadro B.5: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones, Respuestas y Retweets. Segunda mitad

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{1: 2, 2: 1, 13: 1, 5: 2, 9: 1}
Clauset Newman Moore	{2: 1, 3: 1, 5: 1, 7: 1}
Clique Percolation	{1: 5}
Coda	{73: 2, 48: 1, 81: 1, 58: 1, 92: 1, 95: 1}
Conga	{1: 1, 2: 2, 3: 1, 6: 1, 10: 1, 22: 1}
Congo	{1: 1, 2: 2, 3: 1, 4: 1, 5: 1, 10: 1}
Fastgreedy	{2: 1, 3: 1, 4: 1, 5: 2, 6: 1, 7: 1}
Infomap	{1: 4, 3: 1, 4: 1, 5: 1}
Label Propagation	{1: 6, 2: 1}
Leading Eigenvector	{2: 2, 3: 1, 4: 2, 5: 2}
Multilevel	{2: 2, 4: 1, 5: 1, 6: 2, 7: 1}
Radicchi Strong	{1: 7}
Radicchi Weak	{1: 2, 2: 1, 7: 1, 8: 1, 9: 1, 10: 1}
Spinglass	{3: 1, 4: 1, 5: 4, 7: 1}
Walktrap	{1: 1, 3: 2, 65: 1, 8: 1, 7: 1, 15: 1}

Cuadro B.6: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de *Follows*

B.2. Segunda ejecución

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 11, 1: 11, 2: 1, 5: 1}
Clauset Newman Moore	{1: 12, 2: 2, 3: 1, 5: 1, 7: 1, 14: 1}
Clique Percolation	{0: 4, 1: 6, 2: 2, 13: 1, 9: 1}
Coda	{2: 1, 35: 1, 4: 1, 5: 3, 6: 2, 7: 2, 8: 1, 41: 1, 10: 1, 12: 2, 14: 3, 18: 2, 19: 2, 24: 1, 27: 1}
Conga	{32: 1, 2: 4, 3: 4, 4: 3, 8: 2, 10: 1, 34: 1}
Congo	{2: 4, 3: 4, 4: 3, 8: 2, 10: 1, 34: 1, 30: 1}
Fastgreedy	{1: 9, 2: 10, 3: 2, 5: 1, 13: 1, 14: 1}
Infomap	{1: 22, 14: 1, 15: 1}
Label Propagation	{1: 22, 2: 2}
Leading Eigenvector	{8: 1, 1: 11, 2: 10, 3: 2}
Multilevel	{1: 9, 2: 8, 3: 4, 5: 1, 13: 1, 14: 1}
Radicchi Strong	{1: 24}
Radicchi Weak	{1: 19, 2: 2, 4: 1, 21: 1, 14: 1}
Spinglass	{1: 5, 2: 10, 3: 4, 4: 2, 5: 1, 13: 2}
Walktrap	{1: 11, 2: 8, 3: 2, 5: 1, 12: 1, 14: 1}

Cuadro B.7: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 36, 1: 24, 2: 2, 35: 1, 5: 1, 9: 1, 12: 1, 34: 1, 3: 2, 21: 1, 29: 1}
Clauset Newman Moore	{1: 28, 2: 16, 3: 7, 4: 7, 9: 2, 16: 1, 25: 1}
Clique Percolation	{0: 22, 1: 7, 2: 6, 3: 5, 4: 1, 6: 1, 8: 1, 15: 1, 24: 1}
Coda	{3: 1, 4: 5, 5: 5, 6: 4, 7: 8, 8: 4, 9: 5, 10: 5, 11: 4, 12: 4, 13: 6, 14: 2, 15: 1, 16: 2, 17: 1, 18: 1, 19: 2, 20: 3, 22: 2, 23: 1, 27: 1, 28: 1, 31: 1, 35: 1, 46: 1, 56: 1, 63: 1}
Conga	{2: 3, 3: 3, 4: 2, 5: 3, 6: 2, 7: 3, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 2, 14: 3, 15: 2, 20: 1, 21: 1, 29: 1, 31: 2, 34: 1, 37: 2, 41: 1, 69: 1, 91: 1, 121: 1}
Congo	{2: 3, 3: 3, 4: 2, 5: 3, 6: 2, 7: 3, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 2, 14: 3, 15: 2, 20: 1, 21: 1, 29: 1, 31: 2, 34: 1, 37: 2, 41: 1, 69: 1, 89: 1, 121: 1}
Fastgreedy	{1: 34, 2: 16, 3: 8, 4: 9, 5: 1, 7: 1, 9: 2, 16: 1, 25: 1}
Infomap	{1: 62, 2: 1, 3: 2, 4: 2, 5: 1, 7: 1, 9: 2, 19: 1, 25: 1}
Label Propagation	{1: 65, 2: 6, 3: 1, 4: 1}
Leading Eigenvector	{1: 43, 2: 22, 3: 6, 4: 2}
Multilevel	{1: 34, 2: 16, 3: 8, 4: 9, 5: 1, 7: 1, 9: 2, 16: 1, 25: 1}
Radicchi Strong	{1: 73}
Radicchi Weak	{1: 56, 2: 4, 4: 5, 6: 3, 8: 1, 12: 1, 46: 1, 16: 1, 22: 1}
Spinglass	{1: 6, 2: 29, 3: 12, 4: 12, 5: 1, 6: 2, 7: 4, 8: 2, 9: 1, 10: 1, 11: 1, 16: 1, 19: 1}
Walktrap	{1: 34, 2: 16, 3: 8, 4: 9, 5: 1, 7: 1, 9: 2, 17: 1, 25: 1}

Cuadro B.8: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Respuestas

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 89, 1: 39, 2: 8, 3: 1, 39: 2, 72: 1, 9: 1, 12: 1, 13: 1, 45: 1, 119: 1, 29: 1}
Clauset Newman Moore	{0: 3, 1: 67, 2: 47, 3: 5, 4: 1, 5: 3}
Clique Percolation	{0: 63, 1: 33, 2: 1, 3: 1, 4: 1}
Coda	{1: 1, 2: 1, 3: 2, 4: 13, 5: 17, 6: 14, 7: 20, 8: 6, 9: 3, 10: 6, 11: 5, 12: 8, 13: 2, 14: 4, 15: 3, 16: 4, 17: 2, 18: 1, 19: 1, 20: 3, 21: 1, 22: 1, 23: 3, 24: 1, 25: 1, 26: 3, 27: 2, 28: 1, 29: 2, 30: 1, 32: 1, 33: 1, 37: 1, 38: 1, 40: 1, 41: 4, 43: 1, 45: 1, 48: 1, 49: 1, 57: 1}
Conga	{2: 6, 3: 8, 4: 3, 5: 4, 6: 2, 7: 3, 8: 3, 10: 2, 11: 1, 12: 2, 13: 2, 14: 2, 15: 1, 19: 2, 20: 1, 21: 2, 22: 2, 27: 1, 31: 2, 32: 2, 162: 1, 44: 1, 63: 1, 67: 1, 72: 1, 87: 1, 102: 2, 114: 1}
Congo	{2: 6, 3: 8, 4: 3, 5: 4, 6: 1, 7: 5, 8: 3, 10: 2, 11: 1, 12: 2, 13: 1, 14: 2, 15: 1, 19: 2, 20: 1, 21: 2, 22: 2, 27: 2, 30: 1, 31: 1, 32: 1, 162: 1, 44: 1, 63: 1, 68: 1, 72: 1, 87: 1, 102: 2, 114: 1}
Fastgreedy	{1: 86, 2: 50, 3: 6, 4: 1, 5: 3}
Infomap	{1: 126, 2: 10, 3: 8, 5: 1, 7: 1}
Label Propagation	{1: 138, 2: 7, 3: 1}
Leading Eigenvector	{1: 91, 2: 49, 3: 2, 4: 1, 5: 3}
Multilevel	{1: 86, 2: 50, 3: 6, 4: 1, 5: 2, 6: 1}
Radicchi Strong	{1: 146}
Radicchi Weak	{1: 108, 2: 32, 4: 4, 5: 1, 6: 1}
Spinglass	{1: 24, 2: 63, 3: 14, 4: 7, 5: 6, 6: 4, 7: 3, 8: 5, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 2, 16: 1, 17: 2, 21: 1, 22: 1, 23: 6, 24: 1, 25: 1}
Walktrap	{1: 91, 2: 45, 3: 5, 4: 2, 5: 2, 12: 1}

Cuadro B.9: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de *Retweets*

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Bigclam	{0: 82, 1: 81, 2: 14, 3: 3, 4: 3, 5: 6, 6: 2, 7: 1, 8: 1, 9: 1, 10: 4, 11: 1, 12: 1, 13: 1, 19: 1, 30: 1}
Clauset Newman Moore	{0: 6, 1: 47, 2: 60, 3: 21, 68: 1, 5: 4, 6: 3, 7: 2, 8: 5, 9: 1, 10: 2, 11: 1, 12: 1, 15: 1, 33: 1, 17: 1, 20: 2, 55: 1, 4: 13, 26: 1, 27: 1}
Clique Percolation	{0: 47, 1: 55, 2: 16, 3: 8, 4: 2, 5: 2, 6: 3, 7: 4, 8: 1, 9: 1, 10: 1, 11: 1, 16: 1, 19: 1, 32: 1, 54: 1, 23: 1, 25: 1}
Coda	{2: 3, 3: 5, 4: 5, 5: 21, 6: 17, 7: 18, 8: 8, 9: 9, 10: 12, 11: 8, 12: 15, 13: 4, 14: 4, 15: 7, 16: 4, 17: 5, 18: 4, 19: 3, 20: 5, 21: 1, 22: 2, 23: 5, 24: 2, 25: 3, 26: 5, 27: 1, 28: 1, 30: 1, 33: 1, 35: 1, 36: 1, 37: 1, 39: 1, 40: 1, 43: 5, 45: 1, 47: 1, 48: 3, 50: 2, 52: 1, 54: 1, 57: 1, 61: 1, 63: 1, 64: 1, 87: 1}
Conga	{1: 1, 2: 27, 3: 28, 4: 12, 5: 7, 6: 4, 7: 4, 8: 4, 9: 2, 10: 2, 139: 1, 12: 2, 13: 2, 14: 2, 15: 3, 16: 1, 17: 2, 19: 2, 20: 3, 149: 1, 22: 1, 23: 1, 24: 1, 25: 3, 26: 1, 32: 2, 33: 2, 34: 1, 35: 2, 165: 1, 169: 1, 42: 1, 45: 1, 46: 1, 47: 1, 151: 1, 66: 1, 11: 1, 69: 1, 71: 1, 73: 1, 141: 1, 85: 1, 143: 1, 92: 1, 96: 2, 144: 1, 231: 1, 116: 1, 121: 1, 21: 3}
Congo	{128: 1, 1: 1, 2: 27, 3: 28, 4: 13, 5: 7, 6: 5, 7: 4, 8: 3, 9: 2, 10: 3, 151: 1, 12: 3, 141: 1, 14: 1, 15: 3, 144: 1, 17: 2, 19: 2, 20: 3, 21: 2, 22: 1, 23: 1, 24: 2, 25: 2, 26: 2, 30: 1, 32: 1, 33: 2, 34: 1, 35: 1, 165: 1, 169: 1, 42: 1, 45: 1, 46: 1, 47: 1, 138: 1, 66: 1, 68: 1, 72: 1, 73: 1, 13: 2, 85: 1, 92: 1, 96: 2, 16: 1, 230: 1, 116: 1, 121: 1, 149: 1}
Fastgreedy	{1: 57, 2: 70, 3: 29, 4: 15, 5: 6, 6: 3, 7: 3, 8: 5, 9: 2, 10: 1, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 2, 55: 1, 68: 1, 33: 1, 28: 1, 26: 1}
Infomap	{1: 156, 2: 6, 3: 7, 4: 6, 5: 2, 6: 2, 7: 2, 8: 5, 9: 1, 10: 1, 11: 3, 12: 2, 15: 1, 16: 1, 17: 1, 20: 1, 22: 1, 30: 1, 33: 1, 36: 1, 55: 1, 78: 1}
Label Propagation	{1: 183, 2: 17, 3: 2, 4: 1}

Cuadro B.10: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones, Respuestas y *Retweets*. Primera mitad

Algoritmo	**Nº comunidades: Nº egos con esas comunidades**
Leading Eigenvector	{1: 76, 2: 83, 3: 28, 4: 15, 5: 1}
Multilevel	{1: 57, 2: 67, 3: 28, 4: 18, 5: 7, 6: 3, 7: 2, 8: 5, 9: 3, 10: 1, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 2, 33: 1, 55: 1, 68: 1, 26: 1, 28: 1}
Radicchi Strong	{1: 203}
Radicchi Weak	{1: 113, 2: 36, 4: 17, 6: 9, 7: 1, 8: 5, 10: 3, 12: 2, 14: 3, 16: 2, 18: 1, 19: 1, 21: 1, 23: 1, 28: 1, 30: 1, 32: 1, 38: 1, 50: 1, 57: 1, 89: 1, 92: 1}
Spinglass	{1: 20, 2: 79, 3: 38, 4: 21, 5: 10, 6: 5, 7: 3, 8: 4, 9: 4, 12: 2, 13: 1, 14: 1, 16: 2, 17: 1, 18: 4, 20: 1, 21: 3, 23: 1, 25: 3}
Walktrap	{1: 64, 2: 58, 3: 25, 4: 17, 5: 7, 6: 5, 7: 4, 8: 6, 9: 2, 10: 3, 11: 1, 12: 1, 15: 1, 16: 1, 17: 1, 20: 1, 21: 1, 27: 1, 30: 1, 33: 1, 55: 1, 74: 1}

Cuadro B.11: Se muestra el número de comunidades en las que se ha dividido una comunidad: número de ego con esa cantidad de comunidades según cada algoritmo. Conjunto de datos de Menciones, Respuestas y *Retweets*. Segunda mitad